MICROFICHE APPENDIX

METHOD AND SYSTEM FOR GENERATING STATISTICALLY-BASED MEDICAL PROVIDER UTILIZATION PROFILES

By inventors Jerry G. Seare, M.D., Patricia Smith-Wilson, Kurt VanWagoner, Jean Mattey, Eileen Snyder, Candace Wahlstrom, Michelle Willis, Matthew Bentley, Steve Wenzbauer, Rod Fredette and Vicki Sennett

Medicode, Inc.
5225 Wiley Post Way, Suite 500
Salt Lake City, Utah 84116
(801) 536-1005

The materials which follow, including computer source code and file layouts, are provided in the interest of full disclosure and are illustrative of one preferred embodiment of the invention entitled "METHOD AND SYSTEM FOR GENERATING STATISTICALLY-BASED MEDICAL PROVIDER UTILIZATION PROFILES." Numerous other embodiments of the invention and the inventive concept may include materials which differ from the materials provided herein. Each of those other embodiments of the invention and the inventive concept are intended to be comprehended within the scope of the patent claims of the present invention, and the materials provided herein are not intended to limit the scope of the present invention.

```
#--------------------------------------------------------------------------
#       Module Name :   Pp_comp.4gl
#       Version.Edit:   1.0
#       Date Written:   04/01/94
#       Written By  :   Rodney R. Fredette
#
#       Description: This program counts the number of patients and EOCs
#                    within a user specified index code.  Also counts the
#                    number of patients with complicating factors and the
#                    number of EOCs with complicating factors.
#
#       Edit History:
# Edit     Date      By    Reason
# ----     ----      ---   ------
#   1     4/7/94     rrf   add logic to create a temp index table which
#                          combines the data from the index_detail table
#                          with associated data from the index_global
#                          table to form a new temporary index_detail
#                          table with all necessary data.  Also populate
#                          the new EOC table with each occurence of an EOC
#                          as determined by this programs logic.
#
#   2     5/25/94    rrf   change logic for building TEMP_DATA table which
#                          which holds all detail for related codes.  Now
#                          only retrieve detail data for patients who have
#                          at least one icd9 code in the index (indicator =
#                          "I" in the TMP_INDEX table.
#--------------------------------------------------------------------------
#
database eds
#
globals
   define
      q_text      char(400),
      quote         char(1),
      pd            record like gendbs:prtdev.*,
      #
      # the following are user supplied variable.
      #
      ir            record      # ir = input record, data from user
                    index            like index_detail.index,
                    ok_yn          char(1) #Is data input correct?
                  end record,
       init_flag   smallint,
      q_text      char(400),
      quote         char(1)
end globals

main
   define
      i            record
                  indicator      like index_detail.indicator
                  end record,
      l            record
                  date_of_serv   like e_line.date_of_serv,
                  pos            like e_line.pos,
                  tos            like e_line.tos,
                  proc            like e_line.cpt,
                  mod_1            like e_line.mod_1,
```

```
            icd1            like e_line.icd1,
            charge          like e_line.charge
            end record,
    c        record like e_claim.*,
    q        record
            patient          like e_claim.patient,
            relationship     like e_claim.relationship,
            sex              like e_claim.sex,
            date_of_serv     like e_line.date_of_serv,
            cpt              like e_line.cpt,
            icd1             like e_line.icd1,
            category          like category.category
            end record,
    new_cat          like category.category,
    eoc_profile       like qual_master.profile,
    prev_eoc,
    cur_eoc_num       integer,
    ces_rdate       date,
    new_stat          char(2),
    msg             char(75),
    passed,
    cur_by          smallint,
    ok_flag,
    jcount,
    icount          integer,
    rule_err          char(2),
    prev_pat          like e_claim.patient,
    prev_rel          like e_claim.relationship,
    prev_sex          like e_claim.sex

clear screen
defer interrupt

call startlog("pp_comp.log")

initialize ir.* to null

#
# Check for command line arguments
#
if num_args() >= 1 then
   let ir.index = upshift(arg_val(1))
else
   let msg = 'Pp_comp:Must supply index code on comand line!'
   call errorlog(msg)
   call lSend_mail (get_user(), msg)
   exit program
end if

let msg = "Starting: ", ir.index
call errorlog(msg)

start report to "pp_comp.rpt"

call errorlog ("Creating TMP_INDEX table")
call lMake_index(ir.index)
#
# Do all patient level qualifying checks first.  Determine which
# patients have data for the user specified index, then build a
# temporary patient table containing all detail (line) data needed
# to check qualifying conditions.  Currently, this consists of:
```

```
#      date_of_serv, cpt, icd1
#
# First build temp table (temp_data) containing all data needed
# for patients who have at least one occurence of the main index
#
call errorlog ("building temp_data")

select unique patient, relationship, sex
   from e_line lx, e_claim cx, tmp_index ix
   where lx.e_claim_id = cx.e_claim_id and
         lx.icd1 = ix.icd9 and
         ix.indicator in ("I", "MI") and
         cx.e_claim_id != 0
   into temp tmp_patient

select cx.*, lx.date_of_serv, lx.pos, lx.tos, lx.cpt,
       lx.mod_1, lx.icd1, lx.charge, ix.indicator
   from e_line lx, e_claim cx, tmp_index ix, tmp_patient ip
   where lx.e_claim_id = cx.e_claim_id and
         lx.icd1 = ix.icd9 and
         cx.patient = ip.patient and
         cx.relationship = ip.relationship and
         cx.sex = ip.sex and
         cx.e_claim_id != 0
   into temp temp_data

call errorlog ("creating index")
create index i_td1 on temp_data(patient, relationship, sex)
#
# create yet another temp table to hold the category info, because
# it seems to take too long accessing the CATEGORY table using
# the between clause
#
call errorlog ("Making Cat FILE")

create temp table cat_file (
  cproc         char(5),
   category      char(4))
in ucrspace1 extent size 200;

prepare get_cat1_state from
   "select min(category) from category where ? between beg_cpt and end_cpt"
declare get_cat1 cursor for get_cat1_state

declare ins_cat cursor for
   insert into cat_file values (q.cpt, q.category)
open ins_cat

declare bld_cat cursor for
   select unique cpt from temp_data

let icount = 0
foreach bld_cat into q.cpt
   if int_flag then
      call stop_now()
   end if

   let icount = icount + 1
   if icount mod 100 = 0 then
      let msg = "Cat count=", icount using "<<<,<<&"
```

```
      call errorlog (msg)
    end if

    let q.category = " "
    open get_cat1 using q.cpt
    fetch get_cat1 into q.category
    close get_cat1

    put ins_cat
 end foreach
 close ins_cat

 let msg = "Cat count=", icount using "<<<,<<&"
 call errorlog (msg)

 create unique index i_catf1 on cat_file(proc);

 call errorlog ("Starting Main Process")

 let quote = "\""

 prepare get_cat_state from
   "select category from cat_file where proc = ?"
 declare get_cat cursor for get_cat_state
 #
 # Next scroll thru each patient and use each patients data to fill
 # a temp table (temp_qual) to be used for qualifer checks
 #
 create temp table temp_qual (
     date_of_serv      date,
     cpt               char(5),
     icd1               char(6),
     category           char(4))
 in ucrspace1 extent size 100 next size 100;

 declare upat_curs cursor for
   select patient, relationship, sex, date_of_serv, cpt, icd1
       from temp_data
       order by 1,2,3

 prepare del_temp_data from
    "delete from temp_data where patient = ? and relationship = ? and sex = ?"

 prepare del_qual from "delete from temp_qual"

 declare qual_ins cursor for
    insert into temp_qual values (q.date_of_serv, q.cpt, q.icd1, q.category)
 open qual_ins

 let icount = 0
 let jcount = 0

 call errorlog ("Performing patient qual checks")
 foreach upat_curs into q.*
    if int_flag then
       call stop_now()
    end if

    let q.category = " "
    open get_cat using q.cpt
```

```
   fetch get_cat into q.category

   if icount = 0 then
      let prev_pat = q.patient
      let prev_rel = q.relationship
      let prev_sex = q.sex
   end if

   let icount = icount + 1
   if icount mod 1000 = 0 then
      let msg = "UPAT Detail count=", icount using "<<,<<<,<<&"
      call errorlog (msg)
   end if

   if q.patient != prev_pat or
      q.relationship != prev_rel or
      q.sex != prev_sex then
         let jcount = jcount + 1
         if jcount mod 100 = 0 then
            let msg = "Patient count=", jcount using "<,<<<,<<&"
            call errorlog (msg)
         end if

         close qual_ins

         call qual_check("P") returning passed, eoc_profile, rule_err

         if not passed then
            let msg = "PAT FAIL: ", prev_pat, " - ", prev_rel, " - ",
                  prev_sex, "  Rule: ", rule_err
            call errorlog (msg)
            execute del_temp_data using prev_pat, prev_rel, prev_sex
         end if

         execute del_qual
         open qual_ins

         let prev_pat = q.patient
         let prev_rel = q.relationship
         let prev_sex = q.sex
      end if

      put qual_ins
end foreach
#
# Take care of last patient
#
close qual_ins
call qual_check("P") returning passed, eoc_profile, rule_err
if not passed then
   let msg = "PAT FAIL: ", q.patient, " - ", q.relationship, " - ",
         q.sex, "  Rule: ", rule_err
   call errorlog (msg)
   execute del_temp_data using q.patient, q.relationship, q.sex
end if
execute del_qual

declare ref_curs cursor for
   select * from temp_data
```

```
 let icount = 0
 foreach ref_curs into c.*, l.*, i.*
    if int_flag then
       call stop_now()
    end if

    let icount = icount + 1
    if icount mod 10000 = 0 then
       let msg = "count=", icount using "<<,<<<,<<&"
       call errorlog (msg)
    end if

    let cur_by = year(l.date_of_serv) - c.age      # calc birth year

    output to report r_edit(c.*, l.*, i.*, cur_by)
 end foreach

 let msg = "count=", icount using "<<,<<<,<<&"
 call errorlog (msg)

 finish report r_edit
 #
 # Take care of qualifying conditions that may make currently valid
 # EOC's invalid.  Delete all patient data found with a complicating code
 #
 prepare del_comp_eoc from
  'delete from eoc where e_claim_id = ?"

 call errorlog ("updating Comp Patients")
 declare comp_pat_curs cursor for
  select unique e_claim_id
     from e_claim cc, pat_eoc pe
     where cc.patient = pe.patient and
           cc.relationship = pe.relationship and
           cc.sex = pe.sex

 let icount = 0
 foreach comp_pat_curs into c.e_claim_id
    let icount = icount + 1
    if icount mod 1000 = 0 then
       let msg = "count=", icount using "<<,<<<,<<&"
       call errorlog (msg)
    end if

    execute del_comp_eoc using c.e_claim_id
 end foreach
 let msg = "count=", icount using "<<,<<<,<<&"
 call errorlog (msg)
 call errorlog ("done with comp Patients")
 #
 # Perform EOC qualifier checks on all valid EOCs
 #
 call errorlog ("Performing EOC Qualifier Checks")

 declare qeoc_curs cursor for
    select eoc_num, date_of_serv, proc, icd1
       from eoc
       where index = ir.index and
             eoc_status = "V"
       order by eoc_num
```

```
prepare upd_eoc from
   "update eoc set profile = ? where eoc_num = ?"

open qual_ins
let icount = 0
foreach qeoc_curs into cur_eoc_num, q.date_of_serv, q.cpt, q.icd1
   if int_flag then
      call stop_now()
   end if

   let q.category = " "
   open get_cat using q.cpt
   fetch get_cat into q.category

   if icount = 0 then
      let prev_eoc = cur_eoc_num
   end if

   let icount = icount + 1
   if icount mod 1000 = 0 then
      let msg = "QEOC count=", icount using "<<,<<<,<<&"
      call errorlog (msg)
   end if

   if cur_eoc_num != prev_eoc then
      close qual_ins

      let eoc_profile = " "
      call qual_check("E") returning passed, eoc_profile, rule_err
      execute upd_eoc using eoc_profile, prev_eoc

      execute del_qual
      open qual_ins

      let prev_eoc = cur_eoc_num
   end if

   put qual_ins
end foreach
#
# Take care of last patient
#
close qual_ins

let eoc_profile = " "
call qual_check("E") returning passed, eoc_profile, rule_err
if not passed then
   let msg = "EOC FAIL: ", cur_eoc_num, "  Rule: ", rule_err
   call errorlog (msg)
   let new_stat = rule_err
end if
execute upd_eoc using eoc_profile, cur_eoc_num
#
# Grab the category based on procedure code
#
call errorlog ("Appending Category data")

prepare upd_eoc_cat from
   "update eoc set category = ? where proc = ?"
```

```
    declare cat_curs cursor for
       select unique proc from eoc
          where index = ir.index

    let icount = 0
    let jcount = 0
    foreach cat_curs into l.proc
       let icount = icount + 1
       if icount mod 100 = 0 then
          let msg = "Unique Proc Count: ", icount using "<<,<<&",
             "  New Cat Count: ", jcount using "<<,<<&"
          call errorlog(msg)
       end if

       let new_cat = " "
       open get_cat using l.proc
       fetch get_cat into new_cat
       if status != notfound then
          let jcount = jcount + 1
          execute upd_eoc_cat using new_cat, l.proc
       end if
       close get_cat
    end foreach

    let msg = "Unique Proc Count: ", icount using "<<,<<&",
       "  New Cat Count: ", jcount using "<<,<<&"
    call errorlog(msg)

    let msg = "Done: ", ir.index
    call errorlog (msg)
end main

report r_edit(c, l, i, cur_by)
    define
       i               record
                       indicator   like index_detail.indicator
                       end record,
       l               record
                       date_of_serv   like e_line.date_of_serv,
                       pos            like e_line.pos,
                       tos            like e_line.tos,
                       proc            like e_line.cpt,
                       mod_1            like e_line.mod_1,
                       icd1            like e_line.icd1,
                       charge         like e_line.charge
                       end record,
       c               record like e_claim.*,
       cur_by          smallint,
       cur_eoc_num       integer,
       cur_status    like eoc.eoc_status,
       co_name,
       hdr_line1,
       hdr_text,
       hdr2_text       char(78),
       x1, x2, x3       smallint,
       ascii_val       char(30),
       new_status       like eoc.eoc_status,
       prev_dos         date,
       ok_flag,
```

```
    win_max,                        #size of EOC window
    eoc_cnt_for_pat,
    cur_eoc_is_bad,
    an_eoc_was_bad      smallint,
    eoc_cnt,
    pat_cnt,
    eoc_comp,
    pat_comp,
    grp_tot_eoc_comp    integer

output
  top margin 0
  left margin 0
  bottom margin 0
  page length 66

        # order by c.patient, c.relationship, cur_by, c.sex, l.date_of_serv
  order by c.patient, c.relationship, c.sex, l.date_of_serv

format
  first page header
  let q_text =
     "select count(*) from tmp_index where icd9 = ? and ",
        "indicator = ", quote, "C", quote
  prepare cnt_complic_state from q_text
  declare cnt_complic cursor for cnt_complic_state
  #
  # Get EOC window size for this index
  #
   select beg_win into win_max
     from window
     where staging in
        (select staging from index where index = ir.index)

  if win_max is null or win_max <= 0 then
     call errorlog ("Invalid EOC window")
     exit program
    end if
    #
    # create temporary table to store patients who have at lease one
    # complicating factor.  Later, all the EOC status for this patient will
    # will tbe set to 'CP'
    #
    create temp table pat_eoc (
       patient          char(15),
       relationship     char(1),
       sex              char(1)) in ucrspace1;

    declare ins_pat_eoc cursor for
       insert into pat_eoc values (c.patient, c.relationship, c.sex)
    open ins_pat_eoc

    declare eoc_ins cursor for
       insert into eoc values
       (cur_eoc_num, ir.index, cur_status, " ", i.*, l.*, c.e_claim_id, " ")
    open eoc_ins

    select max(eoc_num) into cur_eoc_num from eoc
    if cur_eoc_num is null or cur_eoc_num <= 0 then
       let cur_eoc_num = 1
```

```
    end if

    let eoc_cnt = 0
    let pat_cnt = 0
    let eoc_comp = 0
    let pat_comp = 0
    let grp_tot_eoc_comp = 0

    let hdr_text = "Care Trends EOC Comparison Report"
    let hdr2_text = "For Index Code: ", ir.index
    let co_name = "MEDICODE, INC."
    let x1 = 41 - (length(co_name) / 2)
    let x2 = 41 - (length(hdr_text) / 2)
    let x3 = 41 - (length(hdr2_text) / 2)
    #
    # Check if I/O device needs to be configured
    #
    let ascii_val = " "
    call parse_ascii(pd.esc_code, "N") returning ok_flag, ascii_val
    if ok_flag then
        print ascii_val
    else
        print
    end if

    print
       column  1,  "Date: ", today using "MM/DD/YY",
       column x1, co_name clipped,
         column 65, "Page: ", pageno using "<,<<#"

    print
       column 01, "Time: ", time,
       column x2, hdr_text clipped

    let hdr_line1 =
       column  1, "pp_comp.4gl",
       column x3, hdr2_text clipped

    print hdr_line1
    skip 5 lines

page header
    print
       column 01, "Date: ", today using "MM/DD/YY",
       column x1, co_name clipped,
         column 65, "Page: ", pageno using "<<#"

    print
       column 01, "Time: ", time,
       column x2, hdr_text clipped

    print hdr_line1
    skip 5 lines

before group of c.sex
    let pat_cnt = pat_cnt + 1
    let eoc_cnt = eoc_cnt + 1
    let prev_dos = l.date_of_serv
    let cur_eoc_is_bad = false
    let an_eoc_was_bad = false
```

```
      let eoc_cnt_for_pat = 1
      let cur_status = "V"
      let cur_eoc_num = cur_eoc_num + 1
# print "rel= ", c.relationship, "  sex= ", c.sex
      #
      # Take care of the first qualifying condition that may make the patient
      # invalid.  The patient history must contain at least two related codes.
      # if not, then set the US column = "QP" (disQualified Patient).
      # Set ok_flag = false so no EOC logic will be done.
      #
      let ok_flag = true

   on every row
      open cnt_complic using l.icd1
      fetch cnt_complic into ok_flag
      close cnt_complic

      if ok_flag then
         #
         # we have encountered a complicating ICD, but has this EOC
         # already been flagged as bad? If not, then add 1 to the running
         # total of the number of EOC's with complicating factors (EOC_COMP)
         #
         if not cur_eoc_is_bad then
            let eoc_comp = eoc_comp + 1
            let an_eoc_was_bad = true
            let cur_eoc_is_bad = true
            let cur_status = "C"
         end if
      end if
      #
      # Now look for a gap in service dates of 60 or more days.  If one
      # is found then a new EOC is starting.
      #

      if l.date_of_serv - prev_dos >= win_max then
         #
         # new EOC
         #
         let eoc_cnt = eoc_cnt + 1
         let cur_eoc_is_bad = false
         let eoc_cnt_for_pat = eoc_cnt_for_pat + 1
         let cur_eoc_num = cur_eoc_num + 1
         let cur_status =  "V"
      end if

      let prev_dos = l.date_of_serv

      put eoc_ins

   after group of c.sex
      flush eoc_ins

      if an_eoc_was_bad then
         put ins_pat_eoc
         let grp_tot_eoc_comp = grp_tot_eoc_comp + eoc_cnt_for_pat
         let pat_comp = pat_comp + 1
      end if

   on last row
```

```
    close eoc_ins
    close ins_pat_eoc

    print
       column 56, "% of"

    print
       column 10, "Totals:",
       column 34, "Count",
       column 45, "Comp",
       column 56, "Count"

    print
       column 10, "-------",
       column 34, "-----",
       column 45, "----",
       column 56, "-----"

    print
       column 10, "Patient",
       column 30, pat_cnt using "#,###,##&",
       column 40, pat_comp using "#,###,##&",
       column 54, (pat_comp / pat_cnt * 100.0) using "##&.&&%"

    print
       column 10, "EOC",
       column 30, eoc_cnt using "#,###,##&",
       column 40, grp_tot_eoc_comp using "#,###,##&",
       column 54, (grp_tot_eoc_comp / eoc_cnt * 100.0) using "##&.&&%"

    skip 2 lines
    print
       column 10, "EOC Window: ", win_max using "<<&"
end report

function init_qual_sql()
   let quote = "\""

   let q_text =
      "select * from qual_master where index = ", quote, ir.index, quote,
         " and (scope = ", quote, "B", quote, " or scope = ?) ",
         " order by profile desc"
   prepare mast_state from q_text
   declare mast_curs cursor for mast_state

   prepare grp_state from
   "select * from qual_group where rule_group = ? order by rule_type, rule_id"
   declare grp_curs cursor for grp_state
   #
   # Rule type II requires 2 or more occurences of the index range in the
   # pat. history, but they must occur on different DOS.  So group by DOS and
   # if more than one row is returned, then everything is dandy.
   # If cat_cpt is null use ranges for indicator.  Otherwise use
   # specific icd9 code in the column qual_ic.cat_cpt
   #
   # changed 6/15/94 by rrf: no longer prepared here, but within the qual_chk
   # function itself.  The cursor is built based on qual_ic information.
   #

   prepare ic_state from
```

```
      "select * from qual_ic where rule_type = ? and rule_id = ?"
   declare ic_curs cursor for ic_state

   prepare cnt_cat_state from
      "select count(*) from temp_qual where category = ?"
   declare cnt_cat cursor for cnt_cat_state

   prepare cc_state from
      "select * from qual_cc where rule_type = ? and rule_id = ?"
   declare cc_curs cursor for cc_state

   let init_flag = true
end function

function qual_check(in_scope)
   define
      in_scope         char(1),                    #(P)atient or (E)OC
      qm               record like qual_master.*,
      qg               record like qual_group.*,
      qi               record like qual_ic.*,
      qc               record like qual_cc.*,
      cur_dos          date,
      profile_num      like qual_master.profile,
      first_row,                                   # boolean used by II rule
      ok_flag,
      cnt,
      passed,                                      # Data passed Qual checks
      rule_passed      smallint,
      hold_status      integer

   let passed = true
   let profile_num = null

   if init_flag is null or not init_flag then
      call init_qual_sql()
   end if

   initialize qm.* to null
   open mast_curs using in_scope
   fetch mast_curs into qm.*
   let hold_status = status
   while hold_status != notfound
      open grp_curs using qm.rule_group
      fetch grp_curs into qg.*
      while status != notfound
         case
            when qg.rule_type = "II"
               #
               # build select statement based on detail rules then
               # derive count of rows over different DOS
               #
               let q_text =
                  "select date_of_serv, count(*) from temp_qual, tmp_index ",
                     "where icd1 = icd9 "

               let first_row = true

               open ic_curs using qg.rule_type, qg.rule_id
               fetch ic_curs into qi.*
               while status != notfound
```

```
    if fld_is_null(qi.cat_cpt) then
       if first_row then
          let first_row = false
          let q_text = q_text clipped,
              " and (tmp_index.indicator = ",
             quote, qi.indicator, quote
       else
          let q_text = q_text clipped,
              " or tmp_index.indicator = ",
             quote, qi.indicator, quote
       end if
    else
       if first_row then
          let first_row = false
          let q_text = q_text clipped,
              " and (icd1 = ", quote, qi.cat_cpt, quote
       else
          let q_text = q_text clipped,
              " or icd1 = ", quote, qi.cat_cpt, quote
       end if
    end if

    fetch ic_curs into qi.*
end while

let q_text = q_text clipped, ") group by 1"
let cnt = 0

prepare cnt_ind_state from q_text
declare cnt_ind cursor for cnt_ind_state

open cnt_ind
fetch cnt_ind into cur_dos, ok_flag
while status != notfound
   let cnt = cnt + 1
   fetch cnt_ind into cur_dos, ok_flag
end while
close cnt_ind

if cnt >= qg.num_required then
   let rule_passed = true
else
   let rule_passed = false
end if
#
# If the qg.logical is false, then invert the results of
# this rule check, ie, False = true, true = false
#
if qg.logical = "F" then
   if rule_passed then
      let rule_passed = false
   else
      let rule_passed = true
   end if
end if
#
# if rule_passed is false then none of the detail parts
# of the rule passed ('OR' boolean) so the patient fails.
# stop checking.
#
```

```
      if not rule_passed then
         let passed = false
         exit while
      end if
   when qg.rule_type = "IC"
      let rule_passed = false
      let cnt = 0
      open ic_curs using qg.rule_type, qg.rule_id
      fetch ic_curs into qi.*
      while status != notfound
         open cnt_cat using qi.cat_cpt
         fetch cnt_cat into ok_flag
         close cnt_cat

         let cnt = cnt + ok_flag

         if cnt >= qg.num_required then
            let rule_passed = true
            exit while
         end if

         fetch ic_curs into qi.*
      end while
      #
      # If the qg.logical is false, then invert the results of
      # this rule check, ie, False = true, true = false
      #
      if qg.logical = "F" then
         if rule_passed then
            let rule_passed = false
         else
            let rule_passed = true
         end if
      end if
      #
      # if rule_passed is false then none of the detail parts
      # of the rule passed ('OR' boolean) so the patient fails.
      # stop checking.
      #
      if not rule_passed then
         let passed = false
         exit while
      end if
   when qg.rule_type = "CC"
      open cc_curs using qg.rule_type, qg.rule_id
      fetch cc_curs into qc.*
      while status != notfound
         open cnt_cat using qc.cat_cpt1
         fetch cnt_cat into cnt
         close cnt_cat

         if cnt >= 1 then
            open cnt_cat using qc.cat_cpt2
            fetch cnt_cat into cnt
            close cnt_cat

            if cnt < qg.num_required then
               if qg.logical = "T" then
                  let passed = false
               end if
```

```
                else
                   if qg.logical = "F" then
                      let passed = false
                   end if
                end if
             end if

             if not passed then
                exit while
             end if

             fetch cc_curs into qc.*
          end while

          if not passed then
             exit while
          end if
       end case

       fetch grp_curs into qg.*
    end while

    #
    # for EOC checks a pass means that a profile has been assigned so exit
    # for patients, a failure (not pass) means the client has failed a
    # qualifying condition so don't bother checking any others (exit).
    #
    if passed then
       if in_scope = "E" then
          exit while
       end if
    else
       if in_scope = "P" then
          exit while
       end if
    end if

    fetch mast_curs into qm.*
    let hold_status = status
    if not passed then
       if hold_status != notfound then
          let passed = true
       end if
    else
       exit while
    end if
 end while

 let profile_num = qm.profile
 return passed, profile_num, qi.rule_id
end function
```

Source Code
(PPRAM)

```
*****************************************************************************
 MAIN.4GL                                                    00014 - 25626760

 Programmer: Matt Bentley
             Rod Fredette
             Steve Wenzbauer
    Purpose: This is the main module for the pp_ram.4ge executable program.
    Revised: March 7, 1994
             March 16, 1994 rrf added check of gendbs:app_stat table
             March 22, 1994 sjw changed which fields are checked for
                determination of a single bill.
*****************************************************************************)
lobals "ces_globs.4gl"


lefine
    a_s         record like gendbs:app_stat.*,
    log_msg      char(150),
    cpt_cnt      integer                  # count of bad cpt's


ain

  define
    fileptr, offset, s_analyzed, s_bad int,
    filename    char(50),
    fnd          smallint,
    quote         char(1)

  set lock mode to wait

  if num_args() < 1 then
     display "Usage: pp_ram.4ge <filename>"
     exit program
  else
     let filename = arg_val(1)
     let fileptr = lopentext(filename, "r")
     (let fileptr = lopentext("/dev/rmt/tf0", "r"))
     if fileptr < 0 then
        display "ERROR: Bad filename"
        exit program
     end if
  end if

  let log_msg = "/client1/pracparam/",filename clipped,".log"
  call startlog(log_msg)
  call errorlog("Starting: ")
  #
  #   Check app_stat table to determine if a row exists for this filename
  #    If not, then add one, else update stat = "Y".  At each interval as
  #    stated in the app_stat table, check the app_stat table, if STAT="N"
  #  then exit program gracefully.
  #
  select * into a_s.*
     from gendbs:app_stat
     where app = filename
  if status = notfound then
     let a_s.app = filename
     let a_s.stat = "Y"
```

CONFIDENTIAL

```
   let a_s.interval = 1000
   insert into gendbs:app_stat values (a_s.*)
else
   let a_s.stat = "Y"
   update gendbs:app_stat set stat = a_s.stat
      where app = a_s.app
end if

let quote = "\""
let log_msg = "select * from gendbs:app_stat where app = ",
   quote, filename clipped, quote
prepare chk_stat_state from log_msg
declare chk_stat cursor for chk_stat_state

(* Determine what record to start at, if no record exists then start at 1 *)
let log_msg="select rooo,analyzed,badcpt from EDS:LOADSTAT ",
   " where file = '", filename clipped,"'"
prepare get_rooo from log_msg
declare c_get_rooo cursor for get_rooo
open c_get_rooo
fetch c_get_rooo into offset,s_analyzed,s_bad
if status = notfound then
   let offset = 0
   let s_analyzed = 0
   let s_bad = 0
else
   let offset = offset + 1          #1000
end if
close c_get_rooo
free c_get_rooo

let log_msg =
   "update eds:loadstat set (rooo,analyzed, badcpt)=(?,?,?) where file ='",
      filename clipped,"'"
prepare einsstat from log_msg

(*****************************************************************************
* This code added by SJW on 3/7/94.
* Purpose: If the file doesn't already have an entry in the loadstat table
*          the program needs to insert a initialized row for it, which it
*          wasn't previously doing.  If a row does exist, it will set the
*          numbers back to 0.
*****************************************************************************)
let log_msg =
   "select count(*) from EDS:LOADSTAT where FILE='", filename clipped, "'"
prepare chk_loadstat from log_msg
declare c_chk_loadstat cursor for chk_loadstat
open c_chk_loadstat
fetch c_chk_loadstat into fnd
close c_chk_loadstat
free chk_loadstat
free c_chk_loadstat

if ( fnd) then
   execute einsstat using "0","0", "0"
else
   let log_msg = "insert into EDS:LOADSTAT values (?,?,?,?)"
   prepare new_loadstat from log_msg
   execute new_loadstat using filename, "0", "0", "0"
   free new_loadstat
```

```
   end if

   #delete from tb_imp_log where num = 99999

   (* If an offset was specified then skip to that row now *)
   if offset > 0 then
      call skip_to_row( fileptr, offset)
   end if
   call proc_file(fileptr, offset, s_analyzed, s_bad)

   call lclosetext(fileptr)

   if a_s.stat = "N" then
      call errorlog ("Exiting due to change app_stat flag setting!")
   else
      call errorlog("DONE")
   end if
end main



function proc_file(infile, l_offset, l_analyzed, l_bad)

   define
      infile, l_offset, l_analyzed, l_bad   int,
      keep,
      numbytes      smallint,
      num2,
      dumb_cnt,
      cntr          int,
      tmpdate       char(6),
      provtype      char(1),
      prev_rend     char(12),
      prev_pat      char(9),
      prev_date     date,
      prev_relat    char(2),
      prev_sex      char(1),
      is_over,
      prev_age      smallint

   #
   # Create a cursor for the LOS X-walk
   #
   prepare ex_los from "select new_proc from RAM_XW where LOS_PROC=?"
   declare c_los cursor for ex_los

   let log_msg = "select id from eds:member_id where id = ?"
   prepare pmem from log_msg
   declare getmem cursor for pmem
   #
   # Create temp CPT table of all valid codes for quicker access
   #
   select count(*) into cpt_cnt
      from systables
       where tabname = "temp_cpt"

   if not cpt_cnt then
      create table temp_cpt (proc   char(5))
         in tmp_eds extent size 300 next size 100;
```

```
   insert into temp_cpt
      select unique proc from gendbs:tb_proc
         where ((id = 'MED' and rel_date = '11/23/92') or
                (id = 'BED') or
                 (id = 'ANE' and rel_date = '08/01/92') or
                (id = 'BNE') or
                 (id = 'HCP' and rel_date = '08/01/92') or
                (id = 'BCP') or
                (id = 'DEN' and rel_date = '10/02/92') or
                (id = 'BEN'))

   create unique index i_tcidx1 on temp_cpt(proc)
end if

prepare pproc from
   "select proc from temp_cpt where proc = ?"
declare getproc cursor for pproc

let log_msg = "select e_prov_id from eds:e_prov where carrier = ?",
              " and rend_prov =? and bill_prov =? and zip = ? and spec = ?"
prepare pep from log_msg
declare fep cursor for pep

let log_msg = "select e_claim_id from eds:e_claim where patient = ?",
              " and age = ? and sex = ? and subscriber = ?",
              " and relationship = ? and bill_id = ? and e_prov_id = ?"
prepare pec from log_msg
declare fec cursor for pec

prepare iep from "insert into eds:e_prov values (?,?,?,?,?,0)"
prepare iec from "insert into eds:e_claim values (0,?,?,?,?,?,?,?)"
prepare iel from "insert into eds:e_line values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
declare iel cursor for pel
open iel
let mCliNum = 99999
let rep.carrier = 99999
call Load_Usls()

for cntr = 1 to 100
   let ga_keep[cntr] = TRUE
end for

call lgettext(infile,0) returning inline, numbytes
let keep = load_cross()

let num2 = numbytes +1
let cntr = 1 + l_offset
let dumb_cnt  = 0 + l_analyzed
let cpt_cnt = 0 + l_bad
let Onedaycnt = 0
let prev_rend = rep.rend_prov
let prev_pat  = rec.patient
let prev_date = rel.date_of_serv
let prev_relat = rec.relationship
let prev_age  = rec.age
let prev_sex  = rec.sex
let is_over = false

while numbytes > 0
```

```
   let keep = load_cross()

   if rec.patient <> prev_pat or
      rep.rend_prov <> prev_rend or
      rel.date_of_serv <> prev_date or
      rec.age <> prev_age or
      rec.sex <> prev_sex or
      rec.relationship <> prev_relat then

      let dumb_cnt = dumb_cnt + Onedaycnt
      call do_that_analyze_thing()
      call eds_insert(Onedaycnt)
      let is_over = false
      let Onedaycnt = 0

      if a_s.stat = "N" then
         exit while
      end if

      let prev_date = rel.date_of_serv
      let prev_rend = rep.rend_prov
      let prev_pat  = rec.patient
      let prev_relat = rec.relationship
      let prev_age  = rec.age
      let prev_sex  = rec.sex
   end if

   if keep then
      let Onedaycnt = Onedaycnt +1
      if Onedaycnt > 99 then
         let Onedaycnt = 99
         if not is_over then
            let is_over = true
            let log_msg = "Claim over 100 at Row: ",cntr using "<<,<<<,<<&"
            call errorlog(log_msg)
         end if
        else
         call loadarray(Onedaycnt)
      end if
   end if

   call lgettext(infile,num2) returning inline, numbytes
   let cntr = cntr + 1

   if cntr mod a_s.interval = 0 then
      flush iel
      execute einsstat using cntr, dumb_cnt, cpt_cnt
      let log_msg = "Row count: ",cntr using "<<,<<<,<<&" ,
                    " Oneday: ", dumb_cnt using "<<,<<<,<<&",
                    " BadCPT: ", cpt_cnt using "<<,<<<,<<&"
      call errorlog(log_msg)
      #
      # Check app_stat table to see if someone wants us to finish up and
      # also see if a new interval has been requested
      #
      open chk_stat
      fetch chk_stat into a_s.*
      close chk_stat
   end if
end while
```

```
   if Onedaycnt > 0 then
      call do_that_analyze_thing()
      call eds_insert(Onedaycnt)
   end if

   close iel
   execute einsstat using cntr, dumb_cnt, cpt_cnt
   let log_msg = "Row count: ",cntr using "<<,<<<,<<&" ,
                 " Oneday: ", dumb_cnt using "<<,<<<,<<&",
                 " BADCPT: ", cpt_cnt using "<<,<<<,<<&"
   call errorlog(log_msg)

nd function



unction loadarray(counter)
efine counter   smallint

   ##Provider stuff
   let ga_eds[counter].carrier = rep.carrier
   let ga_eds[counter].rend_prov = rep.rend_prov
   let ga_eds[counter].bill_prov = rep.bill_prov
   let ga_eds[counter].zip = rep.zip
   let ga_eds[counter].spec = rep.spec
   ##Claim stuff
   let ga_eds[counter].patient = rec.patient
   let ga_eds[counter].age = rec.age
   let ga_eds[counter].sex = rec.sex
   let ga_eds[counter].subscriber = rec.subscriber
   let ga_eds[counter].relationship = rec.relationship
   let ga_eds[counter].bill_id = rep.rend_prov
   let ga_eds[counter].e_prov_id = rec.e_prov_id
   ##Line item stuff
   let ga_eds[counter].e_claim_id = rel.e_claim_id
   let ga_eds[counter].date_of_serv = rel.date_of_serv
   let ga_eds[counter].pos = rel.pos
   let ga_eds[counter].tos = rel.tos
   let ga_eds[counter].cpt = rel.cpt
   let ga_eds[counter].mod_1 = rel.mod_1
   let ga_eds[counter].mod_2 = rel.mod_2
   let ga_eds[counter].charge = rel.charge
   let ga_eds[counter].allow_amt = rel.allow_amt
   let ga_eds[counter].anes_time = rel.anes_time
   let ga_eds[counter].icd1 = rel.icd1
   let ga_eds[counter].icd2 = rel.icd2
   let ga_eds[counter].icd3 = rel.icd3
   let ga_eds[counter].icd4 = rel.icd4

   ##Oneday stuff
   let Oneday[counter].carrier = ga_eds[counter].patient
   let Oneday[counter].provider = ga_eds[counter].rend_prov
   let Oneday[counter].patient = ga_eds[counter].patient
   let Oneday[counter].bill ,= ga_eds[counter].relationship,
                               ga_eds[counter].sex,
                               ga_eds[counter].age
#    let Oneday[counter].bill = ga_eds[counter].bill_id
   let Oneday[counter].zdate = ga_eds[counter].date_of_serv
```

```
    let Oneday[counter].prov_zip = ga_eds[counter].zip
    let Oneday[counter].pos = ga_eds[counter].pos
    let Oneday[counter].tos = ga_eds[counter].tos
    let Oneday[counter].cpt = ga_eds[counter].cpt
    let Oneday[counter].misc_cost = ga_eds[counter].charge
    let Oneday[counter].cost = ga_eds[counter].allow_amt
    let Oneday[counter].icd1 = ga_eds[counter].icd1
    let Oneday[counter].icd2 = ga_eds[counter].icd2
    let Oneday[counter].icd3 = ga_eds[counter].icd3
    let Oneday[counter].icd4 = ga_eds[counter].icd4
    let Oneday[counter].age = ga_eds[counter].age
    let Oneday[counter].sex = ga_eds[counter].sex
    let Oneday[counter].specialty = ga_eds[counter].spec
    let Oneday[counter].modif = ga_eds[counter].mod_1

end function



function load_cross()

   define
      tmp_cpt      char(5),
      provtype    char(1),
      ret_val      smallint,
      tmpdate      char(6)

   let ret_val = 0
   let rec.bill_id = inline[41,55]
#   let provtype = inline[88]
#   if provtype = "2" or provtype = "P" then
     ## e_prov columns
     let rep.carrier = 99999
     let rep.rend_prov = inline[22,33]
     let rep.bill_prov = inline[22,33]
     let rep.zip = inline[34,38]
     let rep.spec = inline[39,40]
     ## e_claim columns
     let rec.patient = inline[1,9]
     let rec.age = inline[13,15]
     let rec.sex = inline[12]
     let rec.subscriber = " "
     let rec.relationship = inline[10,11]
     ## e_line columns
     let tmpdate = fixdate(inline[16,21])
     let rel.date_of_serv = tmpdate
     let rel.pos = inline[56,57]
     let rel.tos = inline[58,60]
     let rel.cpt = inline[61,65]
     let rel.mod_1 = " "
     let rel.mod_2 = " "
     let rel.charge = inline[66,72]
     let rel.allow_amt = inline[66,72]
     let rel.anes_time = 0
     let rel.icd1 = inline[75,79]
     let rel.icd2 = " "
     let rel.icd3 = " "
     let rel.icd4 = " "
```

```
    open getmem using rec.patient
    fetch getmem into rec.patient
    if status <> 0 then
       {*********************************************************************
       * This code added 5/16/94 by SJW
       * This is to cross walk old LOS codes to current E&M codes so that
       * CES will analyze them properly.
       *********************************************************************}
       open c_los using rel.cpt
       fetch c_los into tmp_cpt
       if not(status) then
          let rel.cpt = tmp_cpt
       end if
       close c_los

       open getproc using rel.cpt
       fetch getproc into rel.cpt
       if status = 0 then
          let ret_val = 1
          call cross_wok()
         else
          let cpt_cnt = cpt_cnt + 1
       end if
       close getproc
    end if

  return ret_val
end function



function fixdate(lcdate)
define lcdate char(6)

  define
     retdate   char(6),
     lmonth   char(2),
     lday      char(2),
     lyear      char(2)

  let lday   = lcdate[5,6]
  let lmonth = lcdate[3,4]
  let lyear  = lcdate[1,2]

  if lyear > "99" or lyear < "00" then
     let lyear = "01"
  end if

  case
  when lmonth = "01" or lmonth = "03" or lmonth = "05" or lmonth = "07" or
     lmonth = "08" or lmonth = "10" or lmonth = "12"

     if lday > "31" or lday < "01" then
        let lday = "31"
     end if
  when lmonth = "04" or lmonth = "06" or lmonth = "09" or lmonth = "11"
     if lday > "30"  or lday < "01" then
        let lday = "30"
     end if
```

```
   when lmonth = "02"
      if lday > "28" or lday < "01" then
         let lday = "28"
      end if
   otherwise
      let lday = "01"
      let lmonth = "01"
   end case

   let retdate = lmonth, lday, lyear

   return retdate
end function



function eds_insert(lrecrd_cnt)
define lrecrd_cnt smallint

   define
      counter      smallint,
      new_prov_id,
      new_claim_id   integer

   let new_prov_id = null
   let new_claim_id = null

   for counter = 1 to lrecrd_cnt
      if ga_keep[counter] then
         open fep using ga_eds[counter].carrier,
                        ga_eds[counter].rend_prov,
                        ga_eds[counter].bill_prov,
                        ga_eds[counter].zip,
                        ga_eds[counter].spec
         fetch fep into new_prov_id                  #ga_eds[counter].e_prov_id
         if status = notfound then
            execute iep using ga_eds[counter].carrier,
                              ga_eds[counter].rend_prov,
                              ga_eds[counter].bill_prov,
                              ga_eds[counter].zip,
                              ga_eds[counter].spec
            let new_prov_id = sqlca.sqlerrd[2]

            execute iec using ga_eds[counter].patient,
                              ga_eds[counter].age,
                              ga_eds[counter].sex,
                              ga_eds[counter].subscriber,
                              ga_eds[counter].relationship,
                              ga_eds[counter].bill_id,
                              new_prov_id

            let new_claim_id = sqlca.sqlerrd[2]
         else
            open fec using ga_eds[counter].patient,
                           ga_eds[counter].age,
                           ga_eds[counter].sex,
                           ga_eds[counter].subscriber,
                           ga_eds[counter].relationship,
                           ga_eds[counter].bill_id,
```

```
                    new_prov_id
         fetch fec into new_claim_id
         if status = notfound then
            execute iec using ga_eds[counter].patient,
                              ga_eds[counter].age,
                              ga_eds[counter].sex,
                              ga_eds[counter].subscriber,
                              ga_eds[counter].relationship,
                              ga_eds[counter].bill_id,
                              new_prov_id
            let new_claim_id = sqlca.sqlerrd[2]
         end if
      end if
      #
      # Just get the prov_id / claim_id for the first keeper, then exit
      #
      exit for
   end if
end for

if new_prov_id is not null then
   for counter = 1 to lrecrd_cnt
      let ga_eds[counter].e_prov_id = new_prov_id
      let ga_eds[counter].e_claim_id = new_claim_id

      if ga_keep[counter] then
         put iel from ga_eds[counter].e_claim_id,
                      ga_eds[counter].date_of_serv,
                      ga_eds[counter].pos,
                      ga_eds[counter].tos,
                      ga_eds[counter].cpt,
                      ga_eds[counter].mod_1,
                      ga_eds[counter].mod_2,
                      ga_eds[counter].charge,
                      ga_eds[counter].allow_amt,
                      ga_eds[counter].anes_time,
                      ga_eds[counter].icd1,
                      ga_eds[counter].icd2,
                      ga_eds[counter].icd3,
                      ga_eds[counter].icd4
      end if
   end for
end if

for counter = 1 to lrecrd_cnt
   if ga_keep[counter] = TRUE then
   end if
end for

nd function



unction sort_this_bill_into_dates_of_service(larray_cnt)
efine larray_cnt   smallint

   define
      i, j   smallint
```

```
  for i = 1 to (larray_cnt -1)
     for j = (i +1) to larray_cnt
        if ga_eds[j].date_of_serv < ga_eds[i].date_of_serv then
           let ga_eds[100].* = ga_eds[i].*
           let ga_eds[i].* = ga_eds[j].*
           let ga_eds[j].* = ga_eds[100].*
        end if
     end for
  end for

end function



function cross_wok()

  if rel.icd1[4] = "." then
     let rel.icd1 = rel.icd1[1,3],rel.icd1[5]
  end if
  if rel.icd2[4] = "." then
     let rel.icd2 = rel.icd2[1,3],rel.icd2[5]
  end if
  if rel.icd3[4] = "." then
     let rel.icd3 = rel.icd3[1,3],rel.icd3[5]
  end if
  if rel.icd4[4] = "." then
     let rel.icd4 = rel.icd4[1,3],rel.icd4[5]
  end if

  case
  when rec.sex = "1"
     let rec.sex = "M"
  when rec.sex = "2"
     let rec.sex = "F"
  when upshift(rec.sex) = "F"
     let rec.sex = "F"
  when upshift(rec.sex) = "M"
     let rec.sex = "M"
  otherwise
     let rec.sex = "O"
  end case

  if rel.tos[1] = "4" then
     let rep.spec = "5"
  end if

  case
  when rel.tos[1] = "4"
     let rep.spec = "5"
  when rel.tos = "2TO"
     let rel.mod_1 = "80"
  end case

  case
  when rel.pos = "00" or rel.pos = "0G" or rel.pos = "53" or rel.pos = "54"
     or rel.pos = "55" or rel.pos = "90" or rel.pos = "9A"
     or rel.pos = "9C" or rel.pos = "9D" or rel.pos = "BD"

     let rel.pos = "7"
```

```
   when rel.pos = "BM" or rel.pos = "BR" or rel.pos = "CO" or rel.pos = "DO"
      or rel.pos = "OO"

      let rel.pos = "7"
   when rel.pos = "10" or rel.pos = "1S" or rel.pos = "1Z" or rel.pos = "51"
      let rel.pos = "3"
   when rel.pos = "20" or rel.pos = "2F" or rel.pos = "2S" or rel.pos = "2Z"
      or rel.pos = "52" or rel.pos = "56" or rel.pos = "BF" or rel.pos = "BO"

      let rel.pos = "8"
   when rel.pos = "30"
      let rel.pos = "1"
   when rel.pos = "3S" or rel.pos = "40" or rel.pos = "4S"
      let rel.pos = "2"
   when rel.pos = "60"
      let rel.pos = "6"
   when rel.pos = "70" or rel.pos = "80" or rel.pos = "BS"
      let rel.pos = "4"
   end case

   case rec.relationship
   when "AC"
      let rec.relationship = "1"
   when "CD"
      let rec.relationship = "2"
   when "CE"
      let rec.relationship = "3"
   when "CH"
      let rec.relationship = "4"
   when "DD"
      let rec.relationship = "5"
   when "EE"
      let rec.relationship = "6"
   when "LD"
      let rec.relationship = "7"
   when "RR"
      let rec.relationship = "8"
   when "SC"
      let rec.relationship = "9"
   when "SD"
      let rec.relationship = "0"
   when "SP"
      let rec.relationship = "A"
   when "SS"
      let rec.relationship = "B"
   when "ST"
      let rec.relationship = "C"
   end case

end function



function skip_to_row( l_file, l_offset)
define
   l_file, l_offset integer

   define
      numbytes, totbytes integer
```

```
   call lgettext( l_file,0) returning inline, numbytes
   let totbytes = numbytes + 1
   let l_offset = l_offset - 1

   while (numbytes > 0) and (l_offset > 0)
      call lgettext( l_file,totbytes) returning inline, numbytes
      let l_offset = l_offset - 1
   end while

end function
{*****************************************************************************
   analyze.4gl

   Programmer: Matt Bentley
               Rod Fredette
               Steve Wenzbauer
      Purpose: This module calls the analyze functions & merges the results.
      Revised: March 3, 1994
               March 22, 1994 sjw Changed the sort routine and ACW portion
                  of the merge function.
*****************************************************************************}
globals
   "ces_globs.4gl"


* Static Variables *)
define
   bigcnt smallint



function do_that_analyze_thing()

   (* Analyze the Bill Here *)
   let bigcnt = Onedaycnt
   call final_bill()
   call order_bill()
   call cesmain()

   (* Fix the bill based on Log file here *)
   call do_that_merge_thing( Oneday[1].provider, Oneday[1].zdate,
      Oneday[1].carrier, Oneday[1].bill)
   let Onedaycnt = bigcnt

end function




function do_that_merge_thing( l_provid, l_dos, l_patient, l_pat_info)
define l_provid char(15)
define l_dos date
define l_patient   char(15)
define l_pat_info char(15)

   define
```

```
    the_log RECORD LIKE tb_imp_log.*,
    the_line, asd_fnd smallint

 declare q_curs cursor for
    select * from TB_IMP_LOG
    where (NUM = mCliNum) and (BILL = l_pat_info) and (ZDATE = l_dos) and
          (CARRIER = l_patient) and (PROVIDER = l_provid)

 foreach q_curs INTO the_log.*
    if bigcnt = 0 then
       exit foreach
    end if
    let the_line = the_log.line

    case the_log.error
    when "REB"
       call insert_me( the_log.trigger, the_log.cost)
    when "SAS"
       let ga_keep[the_line] = FALSE
    when "UUD"
       let ga_keep[the_line] = FALSE
    when "UID"
       let ga_keep[the_line] = FALSE
    when "UED"
       let ga_keep[the_line] = FALSE
    when "UUS"
       let ga_keep[the_line] = FALSE
    when "UIS"
       let ga_keep[the_line] = FALSE
    when "UES"
       let ga_keep[the_line] = FALSE
    when "TRA"
       let ga_keep[the_line] = FALSE
    when "MFD"
       call fix_mfd( the_log.trigger)
    when "ASD"
       let ga_keep[the_line] = FALSE
    when "ACW"
       (* We know that the original line should be thrown out *)
       let ga_keep[the_line] = FALSE

       ##
       ## If we find an ACW flag, we need to see if it's line was subsequently
       ## deleted with an ASD flag.
       ## If so, we won't insert the crosswalked code at all because there
       ## will be no way to track the inserted line when it comes time to
       ## delete it for the ASD.  Got it?
       ##
       select count(*) into asd_fnd from TB_IMP_LOG
       where (NUM = mCliNum) and (BILL = l_pat_info) and (ZDATE = l_dos) and
          (CARRIER = l_patient) and (PROVIDER = l_provid) and
          (ERROR = "ASD") and (LINE = the_line)
       if not( asd_fnd) then
          call insert_me( the_log.driver, the_log.cost)
       end if
    end case
 end foreach

nd function
```

```
function insert_me( l_trigger, l_cost)
define l_trigger like TB_IMP_LOG.TRIGGER
define l_cost like TB_IMP_LOG.COST

   let bigcnt = bigcnt + 1
   if bigcnt <= 100 then
      let ga_keep[bigcnt] = TRUE

      (* Change the Big array here - element bigcnt *)
      let ga_eds[bigcnt].* = ga_eds[1].*

      let ga_eds[bigcnt].allow_amt = abs(l_cost)
      let ga_eds[bigcnt].cpt = l_trigger
   else
      let bigcnt = 0
   end if

end function
```

```
function fix_mfd( l_trigger)
define l_trigger like TB_IMP_LOG.TRIGGER

   define
      mfd_rec array[100] of record
         proc char(5),
         cost int,
         idx smallint
      end record,
      char_max char(2),
      cur_cnt, i, code_cnt, max_allowed integer,
      the_line smallint

   let code_cnt = 0
   select MAXFREQDAY into char_max from V_PROCEDIT
      where (PROC = l_trigger) and (USL = mfd_c)
   let max_allowed = char_max

   (* Get all the lines with the CPT code into the array *)
   for cur_cnt = 1 to bigcnt
      if ga_eds[cur_cnt].cpt = l_trigger then
         let code_cnt = code_cnt + 1
         let mfd_rec[code_cnt].proc = ga_eds[cur_cnt].cpt
         let mfd_rec[code_cnt].cost = ga_eds[cur_cnt].allow_amt
         let mfd_rec[code_cnt].idx = cur_cnt
      end if
   end for

   (* Sort the Array by descending cost *)
   for cur_cnt = 1 to (code_cnt-1)
      for i = (cur_cnt+1) to code_cnt
         if mfd_rec[cur_cnt].cost < mfd_rec[i].cost then
            let mfd_rec[100].* = mfd_rec[cur_cnt].*
```

```
            let mfd_rec[cur_cnt].* = mfd_rec[i].*
            let mfd_rec[i].* = mfd_rec[100].*
         end if
      end for
   end for

   (* Determine which lines should be deleted *)
   for cur_cnt = 1 to code_cnt
      if (cur_cnt > max_allowed) then
         (* Delete this line item *)
         let the_line = mfd_rec[cur_cnt].idx
         let ga_keep[the_line] = FALSE
      end if
   end for

nd function
```

```
*************************************************************************
 final_bill()

 This function will initialize various columns in the current bill.
 Fields that will be initialized: RVU, LINE, UNLISTED, MODCHECK, SEX
*************************************************************************)
nction final_bill()

 define
    theDesc char(48),
    theCnt integer,
    cnt smallint

 for cnt = 1 to Onedaycnt
    let ga_keep[cnt] = TRUE
    let Oneday[cnt].line = cnt

    if lValidcpt( Oneday[cnt].cpt) and Oneday[cnt].cpt[1,3] != "099" then
       if Oneday[cnt].cpt < "02000" then
          let Oneday[cnt].rvu = -2
       else
          select RVU into Oneday[cnt].rvu from V_PROCDESC
             where PROC = Oneday[cnt].cpt
       end if
    else
       let Oneday[cnt].rvu = -1
    end if
 end for

d function
```

```
nction order_bill()

 define cur_cnt, i smallint
```

```
(* Order by RVU desc *)
for cur_cnt = 1 to (Onedaycnt-1)
   for i = (cur_cnt+1) to Onedaycnt
      if (Oneday[cur_cnt].rvu < Oneday[i].rvu) or
         ((Oneday[cur_cnt].rvu = Oneday[i].rvu) and
         (Oneday[cur_cnt].cpt > Oneday[i].cpt)) or
         ((Oneday[cur_cnt].rvu = Oneday[i].rvu) and
         (Oneday[cur_cnt].cpt = Oneday[i].cpt) and
         (Oneday[cur_cnt].cost < Oneday[i].cost)) then

         let Oneday[100].* = Oneday[cur_cnt].*
         let Oneday[cur_cnt].* = Oneday[i].*
         let Oneday[i].* = Oneday[100].*
      end if
   end for
end for
```

M&G No. 12344.2-US-C1

METHOD AND SYSTEM FOR GENERATING STATISTICALLY-BASED MEDICAL PROVIDER UTILIZATION PROFILES

Microfiche Appendix

This specification includes a Microfiche Appendix which includes 1 page of microfiche with a total of 37 frames. The microfiche appendix includes computer source code of one preferred embodiment of the invention. In other embodiments of the invention, the inventive concept may be implemented in other computer code, in computer hardware, in other circuitry, in a combination of these, or otherwise. The Microfiche Appendix is hereby incorporated by reference in its entirety and is considered to be a part of the disclosure of this specification.

I. Background of the Invention

A. Field of the Invention

The invention relates to methods and systems for analyzing medical claims histories and billing patterns to statistically establish treatment utilization patterns for various medical services. Data is validated using statistical and clinically derived methods. Based on historical treatment patterns and a fee schedule, an accurate model of the cost of a specific medical episode can be created. Various treatment patterns for a particular diagnosis can be compared by treatment cost and patient outcome to determine the most effective treatment approach. It is also possible to identify those medical providers who provide treatment that does not fall within the statistically established treatment patterns or profiles.

B. The Background Art

It is desirable to compare claims for reimbursement for medical services against a treatment pattern developed from a large body of accurate medical provider billing history information. Although in the prior art some attempt was made to compare claims for reimbursement for medical services to a normative index, the prior art did not construct the normative index based on actual clinical data. Rather, the prior art based the normative index on a subjective conception (such as the medical consensus of a specialty group) of what the proper or typical course of treatment should

be for a given diagnosis. Such prior art normative indices tended to vary from the reality of medical practice. In the prior art, automated medical claims processing systems, systems for detecting submission of a fraudulent medical claims, and systems for providing a medical baseline for the evaluation of ambulatory medical services were known. Documents which may be relevant to the background of the invention, including documents pertaining to medical reimbursement systems, mechanisms for detecting fraudulent medical claims, and related analytical and processing methods, were known. Examples include: U.S. Pat. No. 4,858,121, entitled "Medical Payment System" and issued in the name Barber et al. on Aug. 15, 1989; U.S. Pat. No. 5,253,164, entitled "System and Method for Detecting Fraudulent Medical Claims Via Examination of Service Codes" and issued in the name of Holloway et al. on Oct. 12, 1993; U.S. Pat. No. 4,803,641, entitled "Basic Expert System Tool" and issued in the name of Hardy et al. on Feb. 7, 1989; U.S. Pat. No. 5,658,370, entitled "Knowledge Engineering Tool" and issued in the name of Erman et al. on Apr. 14, 1987; U.S. Pat. No. 4,667,292, entitled "Medical Reimbursement Computer System" and issued in the name of Mohlenbrock et al. on May 19, 1987; U.S. Pat. No. 4,858,121, entitled "Medical Payment System" and issued in the name of Barber et al. on Aug. 15, 1989; and U.S. Pat. No. 4,987,538, entitled "Automated Processing of Provider Billings" and issued in the name of Johnson et al. on Jan. 22, 1991, each of which is hereby incorporated by reference in its entirety for the material disclosed therein.

Additional examples of documents that may be relevant to the background of the invention are: Leape, "Practice Guidelines and Standards: An Overview," QRB (Feb. 1990); Jollis et al., "Discordance of Databases Designed for Claims Payment versus Clinical Information Systems," Annals of Internal Medicine (Oct. 15, 1993); Freed et al., "Tracking Quality Assurance Activity," American College of Utilization Review Physicians (November, 1988); Roberts et al., "Quality and Cost-Efficiency," American College of Utilization Review Physicians (November, 1988), Rodriguez, "Literature Review," Quality Assurance and Utilization Review - Official Journal of the American College of Medical Quality (Fall 1991); Elden, "The Direction of the Healthcare Marketplace," Journal of the American College of Utilization Review Physicians (August 1989); Rodriguez, "Literature Review," Quality Assurance and

Utilization Review - Official Journal of the American College of Medical Quality (Fall 1991); Roos et al., "Using Administrative Data to Predict Important Health Outcomes," Medical Care (March 1988); Burns et al., "The Use of Continuous Quality Improvement Methods in the Development and Dissemination of Medical Practice Guidelines, QRB (December, 1992); Weingartèn, "The Case for Intensive Dissemination: Adoption of Practice Guidelines in the Coronary Care Unit," QRB (December, 1992); Flagle et al., "AHCPR-NLM Joint Initiative for Health Services Research Information: 1992 Update on OHSRI," QRB (December, 1992); Holzer, "The Advent of Clinical Standards for Professional Liability," QRB (February, 1990); Gottleib et al., "Clinical Practice Guidelines at an HMO: Development and Implementation in a Quality Improvement Model," QRB (February, 1990); Borbas et al., "The Minnesota Clinical Comparison and Assessment Project," QRB (February, 1990); Weiner et al., "Applying Insurance Claims Data to Assess Quality of Care: A Compilation of Potential Indicators," QRB (December, 1990); Wakefield et al., "Overcoming the Barriers to Implementation of TQM/CQI in Hospitals: Myths and Realities," QRB (March, 1993); Donabedian, "The Role of Outcomes in Quality Assessment and Assurance," QRB (November, 1992); Dolan et al., "Using the Analytic Hierarchy Process (AHP) to Develop and Disseminate Guidelines," QRB (December, 1992); Hadorn et al., "An Annotated Algorithm Approach to Clinical Guideline Development," JAMA (Jun. 24, 1992); Falconer et al., "The Critical Path Method in Stroke Rehabilitation: Lessons from an Experiment in Cost Containment and Outcome Improvement," QRB (January, 1993); Reinertsen, "Outcomes Management and Continuous Quality Improvement: The Compass and the Rudder," QRB (January, 1993); Mennemeyer, "Downstream Outcomes: Using Insurance Claims Data to Screen for Errors in Clinical Laboratory Testing," QRB (June, 1991); Iezzoni, "Using Severity Information for Quality Assessment: A Review of Three Cases by Five Severity Measures," QRB (December 1989); Kahn, "Measuring the Clinical Appropriateness of the Use of a Procedure," Medical Care (April, 1988); Wall, "Practice Guidelines: Promise or Panacea?," The Journal of Family Practice (1993); Lawless, "A Managed Care Approach to Outpatient Review," Quality Assurance and Utilization Review - Official Journal of the American College of Utilization Review Physicians (May, 1990); Dragalin et al., "Institutes for Quality:

Prudential's Approach to Outcomes Management for Specialty Procedures," QRB (March, 1990); Chinsky, "Patterns of Treatment Ambulatory Health Care Management, Physician Profiling - The Impact of Physician, Patient, and Market Characteristics On Appropriateness of Physician Practice in the Ambulatory Setting," (Doctoral Dissertation, The University of Michigan, 1991), published by Concurrent Review Concurrent Review Technology, Inc., Shingle Springs, California; "Patterns of Treatment Ambulatory Health Care Management, Implementation Guide," published by Concurrent Review Concurrent Review Technology, Inc., Shingle Springs, California; "Patterns of Treatment Ambulatory Health Care Management, Patterns Processing Model," published by Concurrent Review Concurrent Review Technology, Inc., Shingle Springs, California; Report on Medical Guidelines & Outcome Research, 4 (Feb. 11, 1993); "Practice Guidelines - The Experience of Medical Specialty Societies," United States General Accounting Office Report to Congressional Requestors (GAO/PEMD-91-11 Practice Guideline) (Feb. 21, 1991); "Medicare Intermediary Manual Part 3 - Claims Process," Department of Health and Human Services, Health Care Financing Administration, Transmittal No. 1595 (April 1993); CCH Pulse The Health Care Reform Newsletter (Apr. 19, 1993); Winslow, "Report Card on Quality and Efficiency of HMOs May Provide a Model for Others," The Wall Street Journal; Jencks et al., "Strategies for Reforming Medicare's Physician Payments," The New England Journal of Medicine (Jun. 6, 1985); Solon et al., "Delineating Episodes of Medical Care," A.J.P.H. (March, 1967); Health Care (September, 1986) (the entire issue of Volume 24, Number 9, Supplement); Miller et al., "Physician Charges in the Hospital," Medical Care (July, 1992); Garnick, "Services and Charges by PPO Physicians for PPO and Indemnity Patients," Medical Care (October, 1990); Hurwicz et al., "Care Seeking for Musculoskeletal and Respiratory Episodes in a Medicare Population," Medical Care (November, 1991), Gold, "The Content of Adult Primary Care Episodes," Public Health Reports (January-February, 1982); Welch et al., "Geographic Variations in Expenditures for Physicians' Services in the United States," The New England Journal of Medicine (Mar. 4, 1993); Schneeweiss et al., "Diagnosis Clusters: A New Tool for Analyzing the Content of Ambulatory Medical Care," Medical Care (January, 1983); Showstack, "Episode-of-Care Physician Payment: A Study of Coronary Arter Bypass

Graft Surgery," Inquiry (Winter, 1987); Schappert, "National Ambulatory Medical Survey: 1989 Summary," Vital and Health Statistics, U.S. Department of Health and Human Services, Public Health Service, Centers for Disease Control, National Center for Health Statistics (April, 1992) (DHHS Publication No. [PHS] 92-1771); Graves, "Detailed Diagnoses and Procedures, National Hospital Discharge Survey, 1990," Vital and Health Statistics, U.S. Department of Health and Human Services, Public Health Service, Centers for Disease Control, National Center for Health Statistics (June, 1992) (DHHS Publication No. [PHS] 92-1774); "National Hospital Discharge Survey: Annual Summary, 1990," Vital and Health Statistics, U.S. Department of Health and Human Services, Public Health Service, Centers for Disease Control, National Center for Health Statistics (June, 1992) (DHHS Publication No. [PHS] 92-1773); "Prevalence of Selected Chronic Conditions: United States, 1986-88," Vital and Health Statistics, U.S. Department of Health and Human Services, Public Health Service, Centers for Disease Control, National Center for Health Statistics (February, 1993) (Series 10, No. 182); "Current Estimates From the National Health Interview Survey, 1991," Vital and Health Statistics, U.S. Department of Health and Human Services, Public Health Service, Centers for Disease Control, National Center for Health Statistics (February, 1993) (DHHS Publication No. [PHS] 93-1512); Iezzoni et al., "A Description and Clinical Assessment of the Computerized Severity Index," QRB (February, 1992); Health Care Financing Review, p. 30 (Winter, 1991); Statistical Abstract of the United States (1992); and Health and Prevention Profile - United States (1991) (published by U.S. Department of Health and Human Services, Public Health Service, Centers for Disease Control, National Center for Health Studies), each of which is hereby incorporated by reference in its entirety for the material disclosed therein.

Additional background materials to which the reader is directed for both background and to refer to while studying this specification include: Physicians' Current Procedural Terminology CPT '94, published by American Medical Association, Code it Right Techniques for Accurate Medical Coding, published by Medicode Inc., HCPCS 1994 Medicare's National Level II Codes, published by Medicode Inc., Med-Index ICD 9 CM Fourth Edition 1993, published by Med-Index, each of which is hereby incorporated by reference in its entirety for the material disclosed therein.

II. Summary of the Invention

It is an object to provide a mechanism for assessing medical services utilization patterns. The invention achieves this object by allowing comparison processing to compare an individual treatment or a treatment group against a statistical norm or against a trend.

It is an object of the invention to provide a mechanism for converting raw medical providers' billing data into a database. The invention achieves this object by read, analyze and merge ("RAM") processing coupled with claims edit processing to achieve a reliable, relevant data set.

It is an object of the invention to provide a mechanism for accurately determining an episode of care. The invention achieves this object by providing a sequence of steps which, when performed, yield an episode of care while filtering out irrelevant and inapplicable data.

It is an object of the invention to provide a method for performing a look-up of information, that is, providing a mechanism for gaining access to different parts of the informational tables maintained in the database. This object is achieved by reviewing the referenced tables for specific codes representing specific diagnoses. The codes are verified for accuracy. Then tables are accessed to display selected profiles. Users are then given the opportunity to select profiles for comparison.

It is an object of the invention to provide a method for comparing profiles. This object is achieved by comparing index codes against historical reference information. Discovered information is checked against defined statistical criteria. The process is repeated for each index code and its profiles developed in the history process as many times as necessary to complete the information gathering.

It is an object of the invention to create, maintain and present to the user a variety of report products. These reports are provided either on-line or in a hard copy format. The process of creating, maintaining and presenting these reports is designed to present relevant information in a complete and useful manner.

It is an object of the invention to provide a mechanism for creating a practice parameter database. This object is achieved in the invention by repetitive

episode of care processing and entry of processed episode of care data into a data table until the populated data table becomes the practice parameter database.

III. Brief Description of the Drawings

FIG. 1 depicts steps performed in the method of the invention to establish a practice parameter or utilization profile for a particular diagnosis.

FIG. 2 depicts an episode of care for a single disease.

FIG. 3 depicts an episode of care for concurrent diseases.

FIG. 4 depicts potential outcomes for an episode of care.

FIG. 5 depicts phases of an episode of care.

FIGs. 6-8 depicts processing of data before episode of care processing begins.

FIG. 9 depicts episode of care processing.

FIG. 10 depicts principle elements of the invention and their relationship to each other.

FIG. 11 depicts the process of the preferred embodiment of the Read, Analyze, Merge element of the invention.

FIG. 12 depicts the process of the preferred embodiment of the Episode of Care element of the invention.

FIG. 13 depicts the process of the preferred embodiment of the Look-up element of the invention.

FIG. 14 depicts the process of the preferred embodiment of the Subset Parameter Look-up component of the Look-up element of the invention.

FIG. 15 depicts the process of the preferred embodiment of the Profile Comparison element of the invention.

IV. Detailed Description of the Preferred Embodiment

The invention includes both a system and a method for analyzing healthcare providers' billing patterns, enabling an assessment of medical services utilization patterns. When the invention is employed, it can readily be seen whether a provider or multiple providers are overutilizing or underutilizing services when

compared to a particular historical statistical profile. The statistical profiles of the invention are a statically-derived norms based on clinically-validated data which has been edited to eliminate erroneous or misleading information. The profiles may be derived from geographic provider billing data, national provider billing data, the provider billing data of a particular payor entity (such as an insurance company) or various other real data groupings or sets. Multiple informational tables are used in the database of the preferred embodiment of the invention. These include a Procedure Description Table, ICD-9 Description Table, Index Table, Index Global Table, Index Detail Table, Window Table, Procedure Parameter Table, Category Table, Qualifying Master Table, Specialty Table, Zip/Region Table, Specialty Statistic Table, Age/Gender Statistic Table, Region Statistic Table, Qualifying Index Table, Qualifying Group Table, Category Parameter Table, and Duration Parameter Table. ICD 9 codes or ICD (International Classification of Diseases, generically referred to as a disease classification) codes as they are generally referred to herein are used in the preferred embodiment. In other embodiments of the invention other codes could be used, such as: predecessors or successors to ICD codes or substitutes therefor, such as DSM 3 codes, SNOWMED codes, or any other diagnostic coding schemes. These tables are described in detail as follows. It should be noted, however, that these tables described are used by the inventors in one implementation of the invention, and that the inventive concept described herein may be implemented in a variety of ways.

PROCEDURE DESCRIPTION TABLE

This table identifies and validates five years of both CPT (Current Procedural Terminology, generically referred to as an identifying code for reporting a medical service) and HCPCS level II procedure codes. The lifetime occurrence maximum and follow-up days associated with a procedure code are also located in this table.

Code (Key)	Alpha/Numeric	5	Standard CPT or HCPCS (5 Years including Modifiers)
Sub-Code	Character	2	* = Starred Procedures N = New Codes Current Year D1 = Deleted Code Current Year D2 = Deleted Code Previous Year D3 = Deleted Code Third Year D4 = Deleted Code Fourth Year C = Changed Description
Life Time Occurrence	Numeric	2	Number = Count of occurrence in a lifetime Blank = Not applicable
Follow Up Days	Numeric	3	Number of Follow up Days to procedure.
Description	Character	48	Standard abbreviated description
Total		60	

USE:

- This table can validate CPT and HCPCS codes.
- Five years of codes will be kept.
- Give a brief description of the code.
- Gives the maximum number of occurrences that this code can be done in a lifetime, if applicable. (Programming not addressed, to date)
- Give the number of follow up days to a procedure. (Programming not addressed, to date)
- Modifiers are stored in this table with a "099" prefix (i.e., the 80 modifier is "09980") with a description of the modifier.
- This table interrelates with:
  - Parameter Tables

- Category Table
- Qualifying Tables
- Specialty Table
- CPT Statistic Table

ICD-9 DESCRIPTION TABLE

This table identifies and validates five years of diagnosis codes. It also contains a risk adjustment factor for each diagnosis.

ICD-9 Code (Key)	Alpha/Numeric	5	Left justified, assumed decimal after 3rd position
Sub-Code	Character	2	N = New Code D = Deleted Code C = Changed Code
Indicator	Character	1	* or blank * = code requires 4th and/or 5th digits to be specific
Risk	Alpha/Numeric	2	Overall Classification of Disease
Description	Character	48	Standard abbreviated description

Total 58

USE:

- This table can validate ICD codes.
- Five years of codes will be kept.
- Give a brief description of the code.
- Show if the code is incomplete and in need of a fourth or fifth digit. An ICD code which should have a 4th and/or 5th digit is listed with an "*".
- This file interrelates with:
  - Index Table
  - Index Detail Table
  - Index Global Table
  - Qualifying Master Table
  - Family Table

– All Parameter Tables

INDEX DETAIL TABLE

This table identifies ICD-9 codes relevant to each specific index code and is used to drive the search for each episode of care. ICD-9 codes have been given an indicator which determines whether or not the associated CPT code should be included in the episode of care. Also, an indicator may cause exclusion of any specific patient record from an episode of care analysis.

Index Code	Alpha/Numeric or character	5	Left justified assumed decimal after 3rd position.
Indicator	Character	2	I = Index code R = Related S = signs/symptoms RO = Rule out C = complications (exclude) M = miscoded V = Vcodes MI = Miscoded Index
Beg-ICD	Alpha/Numeric	5	ICD-9 Beginning Range Code
End-ICD	Alpha/Numeric	5	ICD-9 Ending Range Code
Update	Character	1	A, C, or Blank
Total		17	

USE:

- This table drives the search for the Episode of Care (EOC) and is keyed off the Index Code field.
- Other codes to be included in the parameter search are specified in the indicator field.
- ICD codes with an indicator of "C" when found in a patient history will disqualify the entire patient from the EOC process.
- "Some Index Codes are listed in part with "?" and "??" to exhibit that it does not matter what the trailing 4th and/or 5th digit is, the record is to be accessed for the parameter. For example, the Index code may be 701??, meaning that if the first

three digits of the ICD code start with 701 then use the code regardless of what the 4th and/or 5th digit may be."

- ICD codes maintained in this table are listed as complete as verified by the ICD description table, with the exception of ICD codes beginning with an indicator of "M". Programming logic should consider this when using "M" codes in the search process.
- This table is used for drafting and populating a temporary file built from this table and the Index Global Table based on indicators and keys extrapolated from the Index table.

PROGRAM LOGIC TO ASSIGN EPISODE OF CARE

- Any patient history with an ICD from the temp file for the chosen Index code is tagged for possible assignment of Episode of Care.
- Perform a search on patient history for any ICD code from temp file with an indicator of "C". If found, exclude entire patient history from EOC search.
- The qualifying tables are accessed to verify if specific qualifying factors apply to determine if patient history meets criteria for determination of valid episode of care. (See Qualifying Tables for further explanation)
- The qualifying table is then accessed for further delineation of qualifying circumstances by EOC.
- A timeline is tracked, by patient, for all potential Episodes of care that may occur for a given patient history.
- The data is arrayed based on profile classes which are eight subsets of Procedure categories. An aggregate of all procedures can also be reported. (See Category Table for further explanation)
- This table interrelates with:
  - ICD Description Table
  - Index Table
  - Index Global Table
  - Parameter Table

- CPT Statistic Table
- Age/Sex Table

INDEX TABLE

This table provides a preliminary step for assigning and accessing different tables during the Episode of Care process. This table houses the assignment of staging and whether or not the Index Global table should be accessed.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position.
Staging	Character	2	P = preventive A = acute C = chronic L = life threatening M = manifestations
Global Key	Alpha	2	C = complications M1 = miscoded medical vcodes M2 = miscoded surgical vcodes 1= medical vcodes 2 = surgical vcodes
Indicator	Character	2	C = complications V = vcodes
Update	Character	1	A, C, or Blank
Total		12	

USE:

- Once an Index code has been selected, this table is searched for whether or not the global index table needs to be accessed.
- This table assigns the staging for the index code which points to the window table.
- This table interrelates with:
  - ICD Description Table
  - Index Detail Table
  - Index Global Table
  - Window Table

INDEX GLOBAL TABLE

This table gives a listing of ICD-9 codes common to most Index codes for either inclusion in an EOC such as preventive or aftercare, or exclusion of a patient history such as medical complications.

GLOBAL KEY	Alpha/Numeric	2	C = complications M1 = miscoded medical vcodes M2 = miscoded surgical vcodes 1 = medical vcodes 2 = surgical vcodes
ICD Beginning	Alpha/Numeric	5	ICD-9 Beginning range code
ICD Ending	Alpha/Numeric	5	ICD-9 Ending range code
Update	Character	1	A, C, or Blank
Total		13	

USE:

- This table is used to identify a generic V Code or complication ICD code to be used in an EOC search for any Index code.
- It is triggered by the Index table.
- The surgical Vcodes include all M1, M2, 1 and 2's.
- Medical Vcodes include M1 and 1.
- A complication ICD code will negate the use of a patient history from the EOC search.
- A temporary file for the index code is created based on ICDs extrapolated from this table as well as the Index detail table
- This table interrelates with:
  - ICD Description Table
  - Index Table
  - Index Detail Table

WINDOW TABLE

This table contains the time period preceding and following an episode of care that must be present without any services provided to the patient relating to the index code or associated codes. These windows are used to define the beginning and end points of an episode of care. This table is driven from the staging field in the index table.

Index Code	Alpha/ Numeric	5	Left justified assumed decimal after 3rd position
Staging Indicator	Character	2	P = Preventive C = Chronic, A = Acute L = Life threatening, M = Manifestation
Beginning Window	Numeric	3	Time Period for no occurrence of ICD for Index Code
Ending Window	Numeric	3	Time Period for no occurrence of ICD for Index Code
Update	Character	1	A, C, or Blank
Total		9	

USE:

- This table is keyed off of the staging indicator and it tells the program how long of a "Clear Window" is needed on both ends of this EOC for it to be valid.

PROCEDURE PARAMETER TABLE

This table contains the specific CPT codes identified for each index code listed chronologically with associated percentiles, mode, and average.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position
Profile	Alpha/Numeric	2	Mnemonic
Procedure	Alpha/Numeric	5	CPT, HCPCS
timeframe	Alpha/Numeric	3	Mnemonic for timeframe or total
50th percentile	Numeric	4	Beginning percentile range
50th percentile	Numeric	4	ending percentile range
75th percentile	Numeric	4	beginning percentile range
75th percentile	Numeric	4	ending percentile range
95th percentile	Numeric	4	beginning percentile range
95th percentile	Numeric	4	ending percentile range
Mode	Numeric	3	Numeric Count
Count	Numeric	7	Number of EOCs for timeframe
Sum	Numeric	7	Number of services for timeframe
Weighting	Numeric	6	Numeric count, assumed decimal (4.2)
Update	Character	1	A, C, or Blank
Total		63	

USE:

- This table shows which CPTs are historically billed and statistically how often for a Specific Index Code.

CATEGORY PARAMETER TABLE

This table contains a listing of the procedural categories identified for each index code listed chronologically with associated percentiles, mode, and average.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position.
Profile	Alpha/Numeric	2	Mnemonic
Category	Alpha/Numeric	4	category
timeframe	Alpha/Numeric	4	Mnemonic of timeframe or total
50th percentile	Numeric	4	beginning percentile range
50th percentile	Numeric	4	ending percentile range
75th percentile	Numeric	4	beginning percentile range
75th percentile	Numeric	4	ending percentile range
95th percentile	Numeric	4	beginning percentile range
95th percentile	Numeric	4	and ending percentile range
Mode	Numeric	4	Numeric Count, assumed decimal (4.2)
Count	Numeric	7	Number of EOCs for the timeframe
Sum	Numeric	7	Number of services for the timeframe
Update	Character	1	A, C, or Blank
Total		56	

USE:

- This table shows which Procedural Categories are historically billed and statistically how often for a Specific Index Code.

DURATION PARAMETER TABLE

This table contains the EOC duration distribution for a given Index code.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position.
Profile	Alpha/Numeric	2	Mnemonic
50th percentile	Numeric	4	beginning range
50th percentile	Numeric	4	ending range
75th percentile	Numeric	4	beginning range
75th percentile	Numeric	4	ending range
95th percentile	Numeric	4	beginning range
95th percentile	Numeric	4	ending range
Mode	Numeric	3	beginning and ending range
Update	Character	2	A = Add C = Change
Total		36	

USE:

- This table gives access to Statistical information about EOC durations of care for a given index code.
- It interrelates with:
  - Index Detail table
  - Parameter table

CATEGORY TABLE

This Table provides a grouping of CPT codes into categories of similar services.

Category	Alpha/Numeric	4	Mnemonics
Beg-CPT	Alpha/Numeric	5	Beginning CPT Range
End-CPT	Alpha/Numeric	5	Ending CPT Range
Update	Character	1	A, C, or Blank
Total		15	

USE:

- Procedure codes have been categorized according to most likely type of service they may represent. It could be characterized as a sorting mechanism for procedure codes.

  The mnemonic used for this category is as follows:

  E_1 =Major E and M E_2 =Minor E and M

  L_1 =Major Laboratory L_2 =Minor Laboratory

  R_{D1} =Major Diagnostic Radiology R_{D2} =Minor Diagnostic Radiology

  R_{T1} =Major Therapeutic Radiology R_{T2} =Minor Therapeutic Radiology

  O_1 =Major Oncology Radiology O_2 =Minor Oncology Radiology

  M_{D1} =Major Diagnostic Medicine M_{D2} =Minor Diagnostic Medicine

  M_{T1} =Major Therapeutic Medicine M_{T2} =Minor Diagnostic Medicine

  S_{D1} =Major Diagnostic Surgery S_{D2} =Minor Diagnostic Surgery

  S_{T1} =Major Therapeutic Surgery S_{T2} =Minor Therapeutic Surgery

  A_1 =Major Anesthesia A_2 =Minor Anesthesia

  P_1 =Pathology J=Adjunct

- Categories are also used for arraying Episodes of Care into profile classes or can be reported as an aggregate. The subsets of the aggregate are:

  0 Common Profile
  1 Surgery/Radiation/Medicine Profile
  2 Medicine/Radiation Profile
  3 Surgery/Radiation Profile
  4 Surgery/Medicine Profile
  5 Radiation Profile
  6 Medicine Profile
  7 Surgery Profile

- This table interrelates with:
  - Parameter Table

- Qualifying Tables
- Procedure Table

QUALIFYING MASTER TABLE

This table provides a preliminary step for determining qualifying circumstances that may eliminate a patient history for determination of an Episode of Care. It also provides the initial sort of an episode of care for a specific profile class.

Index Code	Alpha/Numeric	5	Left justified, assumed decimal after 3rd position
Scope	Alpha	1	P = Patient E = Episode of Care B = Both
Profile	Alpha/Numeric	2	Mnemonic or Blank
Group	Alpha/Numeric	5	Correlates to group ID in Qualifying Group Table
Update	Character	1	A, C, or Blank
Total		14	

USE:

- Preliminary step in EOC qualifying process.
- This table interrelates with:
  - Index Detail Table
  - Qualifying Group Table

- The Scope field of the Qualifying Master Table outlines which set of qualifying circumstances are to be applied. The values of the Scope field include P=patient level, E=Episode of Care level, or B= both.

- The Profile field is numbered based on the 8 different profiles (also outlined under the category table. If blank, a profile is not relevant. They are as follows:

0. Common Profile

1. Surgery/Medicine/Radiation Profile

2. Medicine/Radiation Profile

3. Surgery/Radiation Profile

4. Surgery/Medicine Profile

5. Radiation Profile

6. Medicine Profile

7. Surgery Profile

- The Group field assigns a 5 byte mnemonic that establishes a set of qualifying rule sets for a given index code. This field keys directly to the Qualifying Group Table. The majority of the groups relate to profile classes.

QUALIFYING GROUP TABLE

This table groups certain qualifying circumstances to aid in an efficient search for data meeting the criteria.

Group	Alpha/Numeric	5	Left justified assumed decimal after 3rd position
Rule Type	Alpha/Numeric	2	II = Index Code specific rule IS = specific ICD code rule IC = multiple ICD to category rule CC = Multiple code rule CS = code specific rule IG = ICD to gender rule IA = ICD to age rule
Logical	Alpha/Numeric	1	T = True F = False (toggle)
Rule Identifier	Alpha/Numeric	1	M = Male F = Female if IG rule type
Number required	numeric	2	number value
Update	Character	1	A, C, or Blank
Total		15	

USE:

- This table groups all rules qualifying EOCs.
- This table interrelates with:
  - Qualifying Index Table
  - Qualifying Code Table
  - Qualifying Master Table
- A rule type (or rule types) is assigned by group delineating if the rule applies to a single or multiple ICD, single or multiple CPT or category or any combination thereof.
- The Rule Type is a mnemonic which assigns a common type of logic that is to be implemented in the search for the qualifying circumstances. It is possible that the same rule type could be associated with many different rule identifiers. The rule type will also point to either the Qualifying Index Table or the Qualifying Code Table.

The following is a listing of the rule types. (One skilled in the art would

understand that additional rule types and associated programming logic may be implemented):

Rule Types associated with Qualifying Index Table:

II This related directly to the Index code only.

IC This rule is for any indicated ICD code associated with the Index code as it relates to a category or procedure.

IS This rule is for a specific indicated ICD code associated with the Index code as it relates to a category or procedure.

IG This rule is for any indicated ICD code associated with the Index code as it relates to age.

Rule Types associated with Qualifying Code Table:

CC This rule is for a specific procedure or category as it relates to another specific procedure or category for any ICD code associated with the Index code.

CS This is for a specific procedure or category as it relates to a specific ICD code associated with the Index code.

- The rule identifier is an assigned mnemonic based on what the rule is to achieve.
- The Logical indicates if the rule is positive or negative (inclusionary or exclusionary)
- The Number Required is a count of the number of occurrences required for the rule to be valid.

QUALIFYING INDEX TABLE

Table houses qualifying circumstances based on presence or non-existence of Specific procedures and/or ICD codes that would qualify or disqualify a patient history in the determination of an Episode of Care.

Rule Type	Alpha/Numeric	2	II = Index Code specific rule IS = specific ICD code rule IC = multiple ICD to category rule IA = ICD to age rule EG = ICD to gender
Rule Identifier	Alpha/Numeric	4	assigned from Qualifying Master Table
Indicator	Alpha/Numeric	2	I = Index code R = Related S = signs/symptoms RO = Rule out M = Miscoded V = Vcodes MI = Miscoded Index or Blank
Code	Alpha/Numeric	5	category, CPT, HCPCS, ICD or blank
Update	Character	1	A, C, or Blank
Total		14	

USE:

- To act as a qualifying mechanism for determining if claims information can be used in the assignment of an EOC
- This table interrelates with:
  - Procedure Table
  - Category Table
  - Qualifying Group Table
  - ICD Description Table
  - Index Detail Table

QUALIFYING CODE TABLE

Table houses qualifying circumstances based on the presence or non-existence of a specific combination of procedure codes that would qualify or disqualify a patient history in the determination of an Episode of Care.

Rule Type	Alpha/Numeric	2	CC = Multiple code rule CS = code specific rule
Rule Identifier	Alpha/Numeric	3	As labeled in Qualifying Master Table
Primary code	Alpha/Numeric	5	CPT, HCPCS or category or ICD
Secondary Code	Alpha/Numeric	5	CPT, HCPCS or category or ICD
Update	Character	1	A, C, or Blank
Total		14	

USE:

- To act as a qualifying mechanism for determining if claims information can be used in the assignment of an EOC.
- This table interrelates with:
  - Procedure Table
  - Category Table
  - Qualifying Group Table

SPECIALTY TABLE

Table provides a listing of medical specialties with an assigned numeric identifier.

Specialty (Key)	Alpha/Numeric	3	Medicare specialty indicator
Beg-CPT	Alpha/Numeric	5	Beginning CPT to include
End-CPT	Alpha/Numeric	5	Ending CPT to include
Update	Character	1	A, C. or Blank
Total		14	

USE:

This table is used to specify which Specialty is most commonly used with which CPT.

A description of the specialty will be in the documentation.

ZIP/REGION TABLE

Table provides a listing of geographical zip codes sorted into 10 regional zones, standard HCFA information.

Region Indicator	Alpha/Numeric	2	Medicares Ten Regions
Beg-Zip Code	Numeric	5	Beginning Zip Code Range
Beg-Zip Code	Numeric	5	Ending Zip Code Range
Update	Character	1	A, C, or Blank

Total 13

USE:

This table is used to specify which Medicare Region to use for the statistic table.

SPECIALTY STATISTIC TABLE

Table provides a listing of medical specialties with an assigned numeric identifier.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position.
Specialty	Alpha/Numeric	3	
Beg-CPT Code	Alpha/Numeric	5	Beginning Range (Service Area)
Beg-CPT Code	Alpha/Numeric	5	Ending Range (Service Area)
Category	Alpha/Numeric	4	Mnemonic
Multiplier	Numeric	6	Implied decimal (4.2)
Update	Character	1	A, C, or Blank

Total 29

USE:

This table is a matrix that is directly tied to the parameter table by the index code. Its purpose is to give a numeric multiplier that is applied to the occurrence field in the parameter table, to vary the parameter by service area and/or sex and/or region. (i.e., if the occurrence is 2 and the multiplier for a specialist is 1.5, the specialist may receive a total of 3.) Multiple multipliers may be applicable to each parameter.

AGE/GENDER STATISTIC TABLE

Table provides a listing of each CPT code for an index code with a numerical factor used to adjust the frequency of each code by age and/or gender specific data analysis.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position.
Age	Alpha/Numeric	2	00-99
Sex	Alpha/Numeric	1	M, F or Blank
Category	Alpha/Numeric	3	Mnemonic
Multiplier	Decimal	6	Implied decimal (4.2)
Update	Character	1	A, C, or Blank
Total		18	

USE:

This table is a matrix that is directly tied to the parameter table by the index code. Its purpose is to give a numeric multiplier that is applied to the occurrence field in the parameter table, to vary the parameter by service area and/or sex and/or region. (i.e. if the occurrence is 2 and the multiplier for a male is 1.5, the male may receive a total of 3.) Multiple multipliers may be applicable to each parameter.

REGION STATISTIC TABLE

Table provides a listing of CPT codes for an index code with a numerical factor used to adjust the frequency of each code by regional data analysis.

Index Code	Alpha/Numeric	5	Left justified assumed decimal after 3rd position.
Region	Alpha/Numeric	2	Medicares Ten Regions
Multiplier	Decimal	6	Implied decimal (4.2)
Update	Character	1	A, C, or Blank
Total		14	

USE:

This table is a matrix that is directly tied to the parameter table by the index code. Its purpose is to give a numeric multiplier that is applied to the occurrence field in the parameter table, to vary the parameter by service area and/or sex and/or

region. (i.e., if the occurrence is 2 and the multiplier for a region is 1.5, the region may receive a total of 3.) Multiple multipliers may be applicable to each parameter.

FILE LAYOUT FOR CLAIMS DATA CONTRIBUTION

We prefer Electronic Media Claims National Standard Format; however, if you are not using EMC the following is our suggested layout. Please include an exact layout of the format you use with your submission. The record layout that follows is for each line item that appears on a claim. The charge (field 19) should be the **non-discounted fee-for service**. There should be no aggregation or fragmentation.

Field Number	Description	Length	Alpha/ Numeric	Comments
1.	Rendering Provider ID	15	A/N	Unique provider identification number or SSN
2.	Billing Provider ID	15	A/N	Unique provider identification number or SSN
3.	Provider Specialty	3	A/N	Supply a List of Specialty codes used
4.	Patient ID	17	A/N	Unique patient ID number or SSN. May be an encrypted or encoded format.
5.	DOB	6	N	Patient Date of Birth MMDDYY
6.	Sex	1	A	M = Male, F = Female
7.	Subscriber ID	25	A/N	Insured's I.D. No., Normally SSN
8.	Relationship	1	N	Patient to Subscriber, 1 = Self, 2 = Spouse, 3 = Dependent
9.	Bill ID	15	A/N	Unique claim/bill identification number
10.	From Date of Service	6	N	MMDDYY
11.	To Date of Service	6	N	MMDDYY
12.	Provider Zip	5	N	Standard 5 digit Zip Code
13.	Place of Service	2	A/N	Supply a list of POS codes used
14.	Type of Service	2	A/N	Supply a list of TOS codes used
15.	Procedure Code	5	N	Submitted CPT or HCPC code
16.	Modifier	2	N	Submitted CPT modifier
17.	2nd Modifier	2	N	If multiple modifiers are submitted, show the second modifier used. Anesthesia Modifiers (P1-P6)

18.	Claim type	3	A/N	Payor Class Code-W/C, HCFA, Medicaid etc.
19.	Charge	5	N	Billed amount, right justified, whole dollars
20.	Allowed Amount	5	N	Right justified, whole dollars
21.	# of days/units	5	N	number of days and/or units
22.	Anesthesia time	3	N	Actual Minutes
23.	ICD1	5	A/N	First diagnostic code attached to procedure
24.	ICD2	5	A/N	Second diagnostic code attached to procedure (Both ICD1 & ICD2 are left justified, assumed decimal after 3rd byte)
25.	ICD3	5	A/N	Third diagnostic code attached to procedure
26.	ICD4	5	A/N	Fourth diagnostic code attached to procedure
27.	Out-patient facility	5	A/N	Outpatient facility/outpatient hospital identifier
28.	Revenue Code	3	N	Revenue center code

ACCEPTABLE MEDIA TYPES

* 9 track tape: 1600 or 6250 BPI, ASCII or EBCDIC, Labeled or Unlabeled, Unpacked data, Fixed record lengths
* Floppy disk; 3.5" (1.44Mb or 720K) or 5.25" (1.2Mb or 360K), Standard MS-DOS formatted disk, ASCII fixed record length or delimited file
* DC 600A or DC 6150 cartridge : "TAR" or single ASCII or EBCDIC file, Unpacked data, Fixed record lengths
* 8 mm Exabyte tape: "TAR" or single ASCII or EBCDIC file, Unpacked data, Fixed record lengths
* 3480 cartridge: Unpacked data, Fixed record lengths, Compressed or Uncompressed
* Maximum Block size 64,280

DATA PROCESSING METHODOLOGY

This invention is a process for analyzing healthcare providers' billing patterns to assess utilization patterns of medical services. The method of the invention incorporates a set of statistically derived and clinically validated episode of care data to be used as a paradigm for analyzing and comparing providers' services for specific

diagnoses or medical conditions. This invention utilizes a series of processes to analyze the client's healthcare claims history to create unique parameters. In its preferred embodiment, the invention is implemented in software. The invention provides the following functions or tools to the client: creation of local profiles, display of profiles and comparison of profiles.

The creation of local profiles function gives the client the ability to develop unique episode of care profiles utilizing their own claims history data. The process for creating these profiles is identical to the process used in the development of the reference profiles.

The display of profiles function provides a look-up capability for information stored in the reference tables or in client generated profile tables. This look-up capability may be displayed on the computer screen or viewed as a hard-copy printout.

The comparison of profiles function provides a comparison between any two profile sources with attention to variance between them. Some examples include comparing client specific profiles to reference tables, comparing a specific subset of the client's data (eg, single provider) against either reference tables or the client's profiles, or comparing different subsets of the client's profiles to subsets of reference tables.

There are four main processes involved in the invention, as depicted in FIG. 10. These are Read, Analyze and Merge (RAM), 1001, further depicted in FIG. 11; Episode of Care analysis (EOC), 1002, further depicted in FIG. 12; Look-up function, 1003, further depicted in FIGS. 13 and 14; and Profile Comparison, 1004, further depicted in FIG. 15. The invention also includes an innovative reporting mechanism. Each of these four main processes and the reporting mechanism is described in detail in the remainder of this section.

A. Transforming Raw Data Into an Informative Database

Both the RAM and the EOC processes involve healthcare claims history search and analysis. The intent of the RAM and the EOC claims history processing is to enable the end user to establish their own unique profiles based on their existing claims data information. Developing a database of historical provider billing data

which will be used to provide the functionality of the invention is the first step in the invention.

1. Read, Analyze and Merge ("RAM")

In order to define a profile a significant quantity of historical medical provider billing information must be analyzed. As indicated above, the provider billings may come from a variety of sources, with the general guideline that accuracy and completeness of the data and a statistically significant sample of provider billings are required to develop a reliable profile. In the preferred embodiment of the invention, no less than two years of consecutive claims history are used to develop the profiles. The RAM process verifies existence and validity of all data elements in a claims history before the data is processed to develop a profile. The reader is directed to FIGS. 1 and 6-8 for pictorial representations of the preferred embodiment of the invention. FIG. 1 depicts the high level steps performed in one embodiment of the invention. The data flow shown in FIG. 1 includes loading client data 101 from tape 100, reordering various fields 103 and performing date of service expansion 104 as necessary. Next, data are merged (combined) 1-5 and sorted 106 to ensure all bill IDs are grouped together. The data 108 are then read, analyzed and merged into an extended data set (EDS) 110. Reporting and any other processing may occur 111 and an Episode of Care database 112 is created. In the preferred embodiment of the invention, the steps of the invention are implemented in a software product referred to as Care Trends available from Medicode, Inc. of Salt Lake City, Utah.

FIG. 6 depicts read, analyze and merge processing that occurs in the preferred embodiment of the invention. First, one claim at a time the data 603 are read 601, crosswalked and scrubbed (filtered) 602. Then a claim is analyzed 604 with the results output to a log file 605. The results in the log file 605 are then compared 606 to the original claim data and inserted 607 into an extended data set 608.

FIG. 7 depicts an analytical process of the preferred embodiment that includes initializing 701 RVU and line number for each line of the claim and sorting 702 by RVU (descending) and CPT and charge in order to prepare for proper analysis by Medicode's Claims Edit System (CES). Then 703 line items are split into two

groupings of surgical assistant modifiers and all other modifiers in separate groups. Each of the two groups is then validated 704 against disease classification codes (ICD 9); procedure edits rules 705 (CES tables) and unbundle/rebundle edits 706 are performed.

FIG. 8 depicts the merge process of the preferred embodiment of the invention. It includes reading 802 each line from the log file for the current bill, proceeding with processing if the record read is pertinent 804 and determining whether to add the record to the extended data set 805-807.

The following text includes a written description of the RAM processing that is performed in the preferred embodiment of the invention. FIG. 11 shows the RAM process.

The first step in the RAM process is determination of a patient record, 1101. It is necessary to establish a patient record that can be used in the episode of care extraction process (explained in detail below). In the preferred embodiment, a patient record is identified as a unique patient history involving no less than two years of sequential claims history. Because identifying patient information is often removed from patient records to ensure patient confidentiality, patient information such as subscriber/relationship, patient ID, age, gender, bill ID and claim ID may be useful in positively identifying a particular patient. It should be noted that claims history data from various sources may need to be handled differently to identify patient records due to differences in file organization and level of detail of information provided. The amount of information desired to be captured may vary in different embodiments of the invention, but generally the information to be captured is that on a standard HCFA 1500 billing form, Electronic Media Claims, UB 82 or UB 92 claim forms, all of which are generally known in the industry.

The next step, 1102, is the manipulation of the client file layout to extrapolate or crosswalk the pertinent information in order to conform to the logic of the invention. Examples of this step include: translation of type of service, specialty type, modifiers, and/or place of service information.

The next steps involve the validation of claims elements. Each line item of claims history is compared against the Procedure, Description tables, (such as CPT or

HCPCS description tables; such tables generally are referred to as Description Tables and may contain any coding schemes) and the ICD description tables to validate the codes contained in the line item, 1103. Line items with an invalid code are not included in the remainder of RAM processing, though they are counted for future reference. Line items which indicate services being performed over a period of more than one day are expanded into numerous line items, one for each service performed, 1104. The services are then each given a unique date of service beginning with the "date of service from" for the first line item and ending with the "date of service to" for the last line item. The last validation step, 1105, is the conversion of old CPT codes to new CPT codes. This step is essential to provide the most accurate statistics relative to physician office and hospital visits (termed Evaluation and Management Services).

The last step of the RAM process is to edit all claims for errors, through an appropriate claims edit tool, 1106. In the preferred embodiment, software known as "CLAIMS EDIT SYSTEM" which is available from Medicode, Inc. located in Salt Lake City, Utah is used to detect and correct any duplicate line items or inappropriately billed services. This results in an appropriately processed set of raw data that is now in a condition for episode of care processing. The reader is directed to the RAM source code in the Microfiche Appendix for all details of this processing performed in the preferred embodiment.

Figure 9 depicts episode of care formation in the preferred embodiment. This processing includes processing the records in the extended data set that relate to the current index code. This relation is determined by the index tables. Then the records are broken into potential episodes of care based on a period of time specified in a window table. Then the episode of care is qualified based on the rules in a qualifying table. Qualifying episodes of care are inserted into the episode of care table.

2. Determination of Episode of Care

The next step in transforming raw data into a useful database is to determine episodes of care for the data that has already undergone RAM processing. In the invention, a database is created which contains profiles for various diagnoses, chronic and otherwise, including complications indicators. Creation of the database depends on accurately defining an episode of care ("EOC") for each diagnosis. An

episode of care is generally considered to be all healthcare services provided to a patient for the diagnosis, treatment, and aftercare of a specific medical condition. The episode of care for a single disease is depicted in FIG. 2. In the simplicity of the figure, it can be seen that for the diagnosis in question, all healthcare services provided between onset and resolution should be incorporated into the database. An example of this would be a patient who has been afflicted with acute appendicitis. The patient's life prior to onset of the acute appendicitis would be considered a disease free state. On some date, the patient would notice symptoms of acute appendicitis (although he may not know the diagnosis) that cause him to seek the attention of a medical provider. That event would be considered the onset. During the disease state, numerous events may occur, such as the patient consulting a family practitioner, consulting a surgeon, laboratory work and surgical services being performed, and follow-up visits with the provider(s). When further follow-up is no longer required, resolution has been reached. Thus an episode of care has been defined and data from that patient's episode of care is used in the invention to construct a profile for the diagnosis applicable to that patient. Without the use of additional logic, however, the use of that definition of an episode of care would result in erroneous data being entered into the EOC database.

For example, in FIG. 3 it can be seen that a patient suffering from a chronic disease who contracts a second disease could be treated both for the chronic disease and for the second disease during the disease state (i.e. between onset and resolution). If all medical provider billing data during the disease state were entered into the database, then the database would contain erroneous historical data for that individual's diagnosis. For example, if a patient who suffers from psoriasis were to be diagnosed with acute appendicitis and received treatment for psoriasis between the time of onset and resolution of his acute appendicitis, then the provider billings would contain both billings for treatment of the psoriasis and the acute appendicitis. Therefore the invention incorporates methods for discerning medical provider billings relevant to a particular diagnosis. Further, the disease state could be the active state of a chronic disease, and resolution could be the disease returning to its inactive state. A method for handling this situation is therefore also provided.

Other alternatives in the course of a disease further complicate accurately defining an episode of care. From FIG. 4 it can be seen that for any particular diagnosis, the outcome could be resolution, as described above, return to the chronic state of a disease, or complication of the disease. For example, if a patient has undergone an appendectomy, the patient may contract an infection following the surgical procedure. Because complications of various types and durations and in varying frequencies are associated with various diagnoses, a method for incorporating the complication data into the statistically-derived practice parameter is intended to be provided in the invention.

FIG. 5 depicts the phases of an episode of care, including the sequence of patient workup, treatment, and eventual solution, return to the chronic state, or complication followed by either resolution or return to the chronic state.

The method for defining an entire episode of care provided in the invention is used to construct a database of EOCs based on billing data that has been filtered to eliminate data irrelevant to the diagnosis which would lead to an erroneous profile. Essential to the determination of an EOC are certain qualifying circumstances. These circumstances are managed through the use of interrelational qualifying tables, to provide a mechanism for sorting patient history for the occurrence of specific procedures or ICD codes that are requisite for an EOC to be valid.

The steps used in the preferred embodiment to determine an episode of care are shown in FIG. 12 and as follows.

a.) Data Sort by Index Code

First, 1201, a temporary file is created based on combining the authorized and/or disallowed ICD codes that are associated with a given index code in the Index Global Table (listing preventative and aftercare codes) and the Index Detail tables. The temporary file is created using the Index Table, which determines whether or not the Index Detail Table only should be accessed or whether the Index Global Table is also necessary for drafting the temporary file. Second 1202, the raw data set which has undergone RAM processing is sorted by index code (i.e. general diagnosis) to find all patient records within a patient history having an occurrence of a particular index code. It is contemplated that the number of occurrences of a particular index code

can be defined by the user. In the present embodiment, it is recommended that the particular index code being sought occur on at least two different dates of service. The valid components of these patient records are then checked against the interrelational qualifying tables to identify ICD codes associated with the chosen index code. The qualifying circumstances identify criteria such as procedures relating to specific medical conditions which may have been indicated as usually requiring an Evaluation and Management (E/M) service during the course of treatment. For example, an occurrence of a qualifying circumstance such as an E/M service during the patient history is considered in the criteria of an episode of care. In addition, the patient records are searched for any complicating ICD codes that would disqualify the patient record for inclusion in the EOC (such as diabetes or renal failure).

Fourth, 1203 the patient records are compared against the interrelational qualifying tables to ensure compliance with all patient-level qualifying rules. Patient records that fail to qualify are no longer considered for EOC evaluation for this Index Code, however, they may still qualify for other Index Code analysis. Fifth, 1205 all relevant line items for every remaining patient record are checked against the temporary file created in step one for complicating diagnosis codes. Any patient record thus identified with a complicating diagnosis code is removed from further EOC processing.

b.) Determination of Clear Windows

Clear window processing defines the onset and resolution points of an episode of care. The actual parameters used in clear window processing may vary in various implementations of the invention. A clear window time period is selected for the specific Index Code from the window table 1206. Next, 1207 proceeding chronologically, each record is compared with the record immediately preceding it. The first record read defines the beginning event of an initial episode of care and the last record read defines the terminating event of a final episode of care. If the two records being compared are separated by a time period equal to, or greater than, the clear window the earlier record is identified as the terminating event of the earlier episode and the later record is identified as the beginning event of the next episode. Accordingly, the initial episode of care and the final episode may be the same episode of care. It is also possible, for the first record and the last record to be the same record.

This iterative process is continued for all remaining records for all patient claims. In this fashion potential EOCs are identified within the patient claims.

c.) Valid Episode of Care

Each potential episode is then checked to determine if the index code in question appears on the required number of dates of service within the EOC 1208. If the index code does not appear the required number of times, the potential EOC is pended. The qualifying tables are then checked to determine if the potential EOC meets the minimum criteria for procedure codes (such as surgical services) that are expected to be found within a potential episode of care for a given index code. If the minimum criteria are not found in an episode of care, it will not be considered in the profile summary. Processing continues for all patient records. Once an EOC has been determined for a set of claims history meeting the criteria for an Index code, a profile is assigned to the EOC based upon different combinations of treatment patterns that are likely to arise for a given medical condition, 1209. There are eight basic profile classes which outline the common combinations of treatment patterns to statistically analyze and store. These Profile Classes are:

0. Common Profile (diagnostic and E/M services common to all of the above).
1. Surgery/Medicine/Radiation Profile
2. Medicine/Radiation Profile
3. Surgery/Radiation Profile
4. Surgery/Medicine Profile
5. Radiation Profile
6. Medicine Profile
7. Surgery Profile
8. Summary Profile (summary of 0-7 above)

After all valid EOCs have been assigned to a unique profile processing continues with population of the procedure and category tables.

d.) Populating the Procedure and Category Parameter Tables

The data from qualified EOCs will be added to the procedure and category parameter tables 1210. Data from all of the episodes of care for each index

code are inserted into the parameter tables to create the summary statistical profiles. In the preferred embodiment these tables are accessed by index code and populated with data from all the episodes of care for each index code to create and provide summary statistics. The procedure description table and category table are also accessed to determine a description of the procedure codes and the service category in which they fall.

The final step of the EOC process is the generation of output reports, 1211. The output report of this step can be either an online look-up report or a hard copy report. Reports are further described below.

The reader is directed to the Microfiche Appendix containing the source code for EOC processing and to FIG. 9 for supplementary information.

B. Use of the Database

1. Look-up Function

In the preferred embodiment of the invention, a look-up function is provided so that various information available in the database may be accessed. In general, a specific diagnosis may be reviewed in each of the tables of the database based on ICD code. In various embodiments of the invention, other look-up functions may be provided based on nearly any category of information contained in the database. In the preferred embodiment of the invention display of profiles is performed as part of the look-up function. Information in the procedure and category parameter tables are displayed by index code sorted chronologically to show a profile.

The specific steps of the preferred embodiment of the Look-Up function of the invention are shown in FIG. 13 and described as follows.

The first step, 1301, is to review the reference tables for a given Index ICD code. Once a specific diagnosis is chosen for review the process moves to step two. In step two, 1302, the ICD description table is accessed to verify that the ICD-9 code is valid, complete and to provide a description of the diagnosis. It will also indicate a risk adjustment factor assigned to the diagnosis.

In step three, the Index tables are accessed, 1303. Next, step four, 1304, is to determine whether or not the chosen ICD code is an Index code. If it is found as an Index code, any additional ICD codes associated which the selected Index code will

be accessed, 1305. If a chosen diagnosis is not listed as an index code, a prompt, 1306, will allow a search for the selected ICD code to list which index code(s) it may be associated with and its indicator, 1307. A word search capability, 1308, is included in the look-up function applicable to the Index code display. A word or words of a diagnosis is entered and a search of possible ICD codes choices would be listed.

The next step, 1309, is to access the Parameter Tables to display selected profiles. The information provided is driven by the index code and is sorted chronologically, by profile class and by category of procedures. The user is then given the opportunity to choose whether the profiles to be accessed are from the reference tables, client developed profiles, or both, 1310. Next the Procedure Description Table, 1311, and the Category Table, 1312, are accessed to ascertain description of procedure codes and categories under which they fall.

The last step of the Look-Up function is the output of report product, 1313. This report may either be on-line look-up process or in the hard copy report format.

The preferred embodiment of the invention also performs subset profile look-up. This permits analysis of profiles based on selected subsets of data such as age, gender, region and provider specialty.

The process for the subset of profiles look-up includes all of the steps necessary for the general profiles look-up and includes the following additional steps shown in FIG. 14 and described below.

The Age/Gender Table is accessed to ascertain the standard age ranges and/or gender selection for a given profile, 1402. This information is stored by index code with an adjustment factor to be multiplied against the occurrence count of each procedure stored in the parameter table. For example, an adjustment factor of 0.6 associated with an age range of 0 to 17 would be calculated against an occurrence count of 10 for CPT code 71021 for Index code 493XX giving an age adjusted occurrence of 6 for that age range.

The Region Statistic Table, 1403, is accessed and used in a similar manner as the Age/Gender Table. This table has adjustment factors based on ten regions throughout the United States.

The Zip/Region Table, 1404, is accessed to identify what region a particular geographic zip code falls within.

The CPT Statistic Table, 1405, is accessed and used in a similar manner as the Age/Gender table. This table has adjustment factors based on different medical specialty groupings.

The Specialty table, 1406, is accessed to ascertain what particular specialty groupings are suggested.

The subset parameter Look-Up function also includes the capability of producing output reports, 1407. These reports can be on-line look-up process reports or hard-copy report format reports.

2. Comparison Processing

In the preferred embodiment of the invention, it is possible to compare profiles developed from a data set against profiles developed from a reference data set. Subsets of profiles may be compared as well. Profiles may be compared for any index code and profile reports may be output. It is also possible to identify those medical providers (whether individuals or institutions) who provide treatment that does not fall within the statistically established treatment patterns or profiles. Further, various treatment patterns for a particular diagnosis can be compared by treatment cost and patient outcome to determine the most effective treatment approach. Based on historical treatment patterns and a fee schedule, an accurate model of the cost of a specific medical episode can be created.

The specific process of Comparison Processing is shown in FIG. 15 and described as follows. The first step, 1501, is the comparison of information developed from the data history search process with reference information stored in the Parameter Tables. The next step, 1502, is to test the services from the history processing to see if it falls within the defined statistical criteria in the Parameter Tables. If it does an indicator is given to this effect, 1504. If the services fall outside the statistical criteria of the reference Parameters Table, a variance alert describing the difference will be given, 1503. The process may be repeated for each index code and its profile developed in the history process, 1505. The final step is to produce output reports, 1506. These reports are either on-line look-up process reports or hard-copy report format reports.

3. Reporting

Reporting of various information contained in the database is provided in the preferred embodiment. Six different types of reports or displays are provided in the preferred embodiment, these are: Provider Practice Profile Report, Profile Comparison Reports, Resident Parameters Display, Local Parameters Display, Parameter Comparison Report and Chronological Forecast. Each of these reports or displays is described as follows.

The Provider Practice Profile Report is a set of reports which provide a tally or summary of total CPT and/or ICD code utilization by a provider or group of providers during a specified time interval and allows comparison against provided reference data or client generated reference data.

The select criteria for running the tally can be any one of the following:

- single physician, department, specialty or clinic by CPT and/or ICD
- multiple physicians, departments, specialties, or clinics by specialty, region, CPT and/or ICD
- period of time being analyzed

Included in the report is the following:

- criteria for select
- claims analyzed
- average lines per bill
- invalid CPTs and percent of total for study
- invalid ICDs and percent of total for study
- incomplete ICDs and percent of total for study
- patients in age categories
- patients by gender
- missing ICDs and percent of total for study

The report includes numerous (up to about 22 in the preferred embodiment) separate procedure (such as CPT) categories which are headers for each page. Each CPT utilized within that category will be reported by:

- frequency and percent of total

- dollar impact and percent of total for single or multiple fee schedules and/or allowable reimbursement schedules
- grand total if more than a single physician report

The report includes a tally by ICD. Each ICD utilized is reported on by:

- frequency and percent of total
- dollar impact and percent of total for single or multiple fee schedule and/or allowable reimbursement schedules (dollar impact based on each line item CPT correlated to the ICD)

If a report includes region and/or specialty, there are numerous tallies for procedure categories and/or ICD.

The Profile Comparison Reports give the client a comparison of a health care provider's (or group of providers') utilization of CPT and/or ICD-9 codes in a specific episode of care against a reference set of utilization profiles. This includes number, frequency and chronological order of services along with other statistical information (eg, range, mode, confidence interval, etc.).

The comparison can be against one of the following:

- national norms resident in the tables
- regional norms resident in the tables
- client established norms developed by use of the tally report, outlined above
- other

Selection criteria include the following:

- single physician, department, clinic or specialty by CPT and/or ICD to be compared against national, regional, specialty, and/or client established norms
- multiple physicians, departments, clinics, or specialties by CPT and/or ICD by specialty and/or region, to be compared against national, region, specialty, and/or client established norms
- set period of time being analyzed

General information included in the report includes:

- criteria for select (i.e., national, regional, specialty, and/or client established)
- claims analyzed
- average lines per bill
- invalid CPTs and percent of total for study and comparison
- invalid ICDs and percent of total for study and comparison
- incomplete ICDs and percent of total for study and comparison
- patients in age categories and comparison
- patients by gender and comparison
- missing ICDs and percent of total for study and comparison

The report includes numerous separate CPT categories which are headers for each page. Each CPT utilized within that category will be reported by:

- frequency and percent of total
- dollar impact and percent of total for single or multiple fee schedules and/or allowable reimbursement schedules
- grand total if more than a single physician report

The report includes a tally by ICD. Each ICD utilized is reported on by:

- frequency and percent of total
- dollar impact and percent of total for single or multiple fee schedule and/or allowable reimbursement schedules (dollar impact based on each line item CPT correlated to the ICD)

If a report includes region and/or specialty, there are numerous tallies for CPT categories and/or ICD.

The Resident Parameters Display provides the client a look-up mode for information stored in the Practice Parameter Tables or client generated parameter tables. This lookup should be on the computer screen or as a print out.

The selection criteria is based on the key elements of the Practice Parameter tables. For Example:

- Index code for associated CPT codes and/or any other the following:
  - index code only

- index code and indicators (i.e, related, complicating, rule/outs, symptoms, etc)
- specialty
- region
- age
- gender
- standard length of Episode of Care
- based on profile (tally)
- based on parameter (timeline)

- regional variables
- other misc. look-ups
  - geozips incorporated in a region
  - CPT for follow up days and/or lifetime occurrence
  - specialty and associated CPT codes
  - ICD and Risk Factor

The Local Parameters Display provides the same information as described in the Display of Resident Parameters listed above.

The Parameter Comparison Reports are a set of reports which give the client a comparison of a physician (or group of physicians) utilization of CPT and/or ICD against an existing set of utilization norms over a timeline and in chronological order.

The comparison can be against one of the following:

- national norms resident in the tables
- regional norms resident in the tables
- client established norms developed by use of the tally report, outlined above
- other

Selection criteria include the following:

- single physician, department, clinic or specialty by CPT and/or ICD to be compared against national, regional, specialty, and/or client established norms

- multiple physicians, departments, clinics, or specialties by CPT and/or ICD by specialty and/or region, to be compared against national, region, specialty, and/or client established norms
- set period of time being analyzed

General information included in the report includes:

- criteria for select (i.e, national, regional, specialty, and/or client established)
- claims analyzed
- average lines per bill
- invalid claims due to incomplete Episode of Care
- invalid CPTs and percent of total for study and comparison
- invalid ICDs and percent of total for study and comparison
- incomplete ICDs and percent of total for study and comparison
- patients in age categories and comparison
- patients by gender and comparison
- missing ICDs and percent of total for study and comparison

The report includes numerous separate procedure categories which are headers for each page. Each procedure category utilized within that category will be reported by:

- frequency and percent of total
- dollar impact and percent of total for single or multiple fee schedules and/or allowable reimbursement schedules
- grand total if more than a single physician report

The Chronological Forecast provides statistical trend analysis and tracking of the utilization of billing codes representative of services performed by a physician for a given diagnosis over a set period of time and stored in chronological order. It will provide a summation of billed codes representative of services and diagnoses utilized by an entity over a period of time.

C. System Requirements

The method and system of this invention may be implemented in conjunction with a general purpose or a special purpose computer system. The

computer system used will typically have a central processing unit, dynamic memory, static memory, mass storage, a command input mechanism (such as a keyboard), a display mechanism (such as a monitor), and an output device (such as a printer). Variations of such a computer system could be used as well. The computer system could be a personal computer, a minicomputer, a mainframe or otherwise. The computer system will typically run an operating system and a program capable of performing the method of the invention. The database will typically be stored on mass storage (such as a hard disk, CD-ROM, worm drive or otherwise). The method of the invention may be implemented in a variety of programming languages such as COBOL, RPG, C, FORTRAN, PASCAL or any other suitable programming language. The computer system may be part of a local area network and/or part of a wide area network.

Referring to Fig. 16 of the drawings and to the Microfiche Appendix, there is illustrated a second embodiment of a method for implementing the present invention for determining episodes of care for a selected medical condition identified by an Index Code. This embodiment is essentially the same as that described above except where noted, and the same nomenclature and tables will be referred to as in the above embodiment. The method is implemented by the computer program module **pp_comp.4gl,** which appears in the Microfiche Appendix.

a) Create Temporary File of ICD-9 Codes Corresponding to Selected Index Code

First, at step 1610, a temporary file, **tmp_index,** is created as a programming convenience, based on the Index Code for which episodes of care are being built. An Index Code identifies a medical condition (e.g., 174? might be the Index Code for the disease, Malignant Neoplasm of Female Breast). In the Index Detail Table, each Index Code is associated with ranges of ICD-9 diagnosis codes relevant to the medical condition, as well as separate Indicator values associated with each range. Only ICD-9 codes with an Indicator value of "I" or "MI" for the associated Index Code are used to drive the creation of an episode of care.

At 1610, the ***pp_comp.4gl*** module, after defining program variables, executes the function, ***lMake_index***, which builds the temporary file, **tmp_index**, that contains a separate record for each ICD-9 code in the ranges of ICD-9 codes associated with the

selected Index Code. (The value of the selected Index Code is passed to ***lMake_index*** in the variable *ir.index*, which contains the Index Code value provided in the input record for ***pp_comp.4gl***, e.g., *index_detail.index*) The function call to the ***lMake_index*** appears at the bottom of page 2 of the **pp_comp.4gl** program listing.

The ***lMake_index*** function creates the **tmp_index** file by extracting from the Index Detail table and the Index Global table information that includes the ranges of ICD-9 codes associated with the selected Index Code and the Indicator value for each of such ICD-9 codes. For example, if, in the Index Detail table, Index Code 174? were associated with the following ranges of ICD-9 codes and Indicator values: 1740 to 1749 for Indicator "I"; 174 for Indicator "MI"; 61172 (Lump Or Mass In Breast) for Indicator "R;" then the **tmp_index** file records correlating to Index Code 174? would include the following information:

Indicator	ICD9
I	1740
I	1741
I	1742
I	1743
I	1744
I	1745
I	1746
I	1747
I	1748
I	1749
MI	174
R	61172

a) Find Patients With Driving ICD-9 Codes

Second, at step 1620, the raw data set that has undergone RAM processing is sorted by ICD-9 code to find all patient records having an occurrence of ICD-9 codes that may drive the creation of an episode of care for the selected Index Code (i.e., ICD-9 codes corresponding to ICD-9 codes in the **tmp_index** file having an Indicator value of "I" or "MI"). More specifically, the **pp_comp.4gl** module first creates a second

temporary file, **tmp_patient**, with the following statement appearing at the top of page 3 of the source code listing.

```
select unique patient, relationship, sex
        from e_line lx, e_claim cx, tmp_index ix
        where ix.e_claim_id = cx.e_claim_id and
                lx.icd1 = ix.icd9 and
                ix.indicator in ("I", "MI") and
                cx.e_claim_id != 0
        into temp tmp_patient
```

This statement creates the temporary table, **tmp_patient**, and populates it with every unique combination of patient, relationship, and sex for every patient record containing an ICD-9 code listed in the **tmp_index** table with an Indicator value of "I", or "MI". Since **tmp_index** table maps Index Codes (medical conditions) to individual ICD-9 codes, the **tmp_patient** table identifies only those patients whose diagnoses in their medical claims history include one of the driving ICD-9 codes for the medical condition in question.

The program then creates a third temporary file, **temp_data,** and populates it with every record from the RAM-processed data set that meets two criteria:

(1) contains a combination of patient, relationship, and sex values that corresponds with a record in the **tmp_patient** table; and

(2) contains an ICD-9 code that corresponds to an ICD-9 code in the **tmp_index** table.

The program statement that implements these two steps appears in the top half of page 3 of the ***pp_comp.4gl*** program listing in the ***select*** statement beginning "**select cx.*, ix.date_of_serv, ...**" and ending with "**into temp temp_data**" Specifically, the following segment of the ***select*** statement links the **e_claim** table, which contains one record for each medical claim identified in the RAM-processed data set, to the

tmp_patient table described above by matching the patient ID number, relationship code, and gender values in the two tables.

from e_line lx, e_claim cx, tmp_index ix, tmp_patient ip

where lx.e_claim_d=cx.e_claim_id and

lx.icd1 = ix.icd9 and

cx.patient = ip.patient and

cx.relationship = ip.relationship and

cx.sex = ip.sex and

...

Next, the following segment of the ***select*** statement links the **e_line** table, which contains all records in the RAM-processed data set (that is, each claim line item that appears in the patients' medical histories), to the **tmp_index** table described above by matching the ICD-9 diagnosis codes in the two tables.

from e_line lx, e_claim cx, tmp_index ix, tmp_patient ip

where lx.e_claim_d=cx.e_claim_id and

lx.icd1 = ix.icd9 and

...

The result of the foregoing two steps is that the **temp_data** table will hold data that meet the following criteria:

1. The claim line items belong to a patient who had an "I" or "MI" somewhere in their medical history.
2. The claim line item includes an ICD-9 code that is also found in the **temp_index** table.

At this point, the **temp_data** table holds claim line items that potentially will be included in an Episode of Care (EOC) for a selected Index Code.

a) Create Procedure Categorization Table

At 1630, the program creates another temporary table, **cat_file** that is used for grouping procedure codes into categories, which are described above in relation to the

Category Table. The categories represent broad classes of treatment or service types, such as Major E and M (Evaluation and Management), Minor E and M, Major Diagnostic Radiology, Minor Diagnostic Radiology, Major Laboratory, and Major Therapeutic Surgery. Categories are used in place of individual procedure codes in subsequent program steps. For example, certain qualifying rules reference category codes rather than individual procedure codes. Also, categories are used to sort episodes of care into profile classes for analysis and reporting purposes.

At step 1630, the program assigns a category mnemonic (e.g., E_1 for Major E and M) to each procedure code found in the **temp_data** file. This program step is implemented by the source code at pages 3-4, beginning with the statement "**call errorlog ("Making Cat File"),**" through the statement, "**create unique index i_catf1 on cat_file(proc);**". Specifically, the **cat_file** table is built by looping through each procedure code in the **temp_data** table, finding every unique CPT/HCPCS code in that table and associating the code found with a category.

b) Check Patient History Against Qualifying Rules

At step 1640, the records from the patient histories (now in the **temp_data** table) are reviewed to ensure compliance with the patient-level qualifying rules defined by the various qualifying tables of the present invention. Patient records that fail to qualify are no longer considered for EOC evaluation for the selected Index Code. The ***pp_comp.4gl*** source code for implementing this step includes the statements beginning at the middle of page 4 with "**declare upat_curs cursor for**" and continuing through the bottom of page 5, "**execute del_qual.**" Pertinent portions of these statements are reproduced below.

```
foreach upat_curs into q.*
...
        call qual_check("P") returning passed, eoc_profile, rule_err
        if not passed then
            ...
            execute del_temp_data using prev_pat, prev_rel, prev_sex
...
```

end foreach

Generally, these program statements perform the following steps:

- read fields from each patient record in the **temp_data** table into *upat_curs*;
- for each patient record in *upat_curs*;
  - ➢ read the record into the variable set *q.**;
  - ➢ call ***qual_check*** function to determine if the patient data on the record satisfies a set of patient qualifying rules, and
  - ➢ if not, remove all of the patient's data from further consideration for the selected Index Code.

These patient qualification steps are repeated until such processing has been completed for all patients having a record in the **temp_data** table.

The Qual_Check Function

The ***qual_check*** function identified above can be found beginning on page 13 of the ***pp_comp.4gl*** program listing, beginning with the statement "**function qual_check(in_scope)**" and continuing through the end of page 16. For the selected Index Code, the ***qual_check*** function loops through all entries in the **qual_master** (Qualifying Master) table where the Scope field is equal to the value passed to the ***qual_check*** function in the *in_scope* variable. (In the present embodiment, the *in_scope* variable is set to either the value "E" or "P", which indicates whether the function checks for 'E'pisode or 'P'atient level qualifying rules.) Here, at step 1640, the value of the *in_scope* variable is set to 'P,' such that only patient level qualifying rules are executed.

Based on the value of the Group field in the Qualifying Master table for the selected Index Code, the ***qual_check*** function extracts qualifying rules information (i.e., Rule Type and Rule Identifier) from the **qual_group** (Qualifying Group) table. More particularly, when the ***qual_check*** function reads a record from the **qual_master** table for the selected Index Code, it uses the value of the *rule_group* field from the **qual_master** record as a parameter to a query for reading a record in the **qual_group**

(Qualifying Group) table. Depending upon the value of the *rule_type* field this **qual_group** table record, the ***qual_check*** function executes a different set of program statements implementing qualifying logic. As will be set forth more fully below, the ***qual_check*** function uses this *rule_type* value to extract information for identifying the proper qualifying rules from either the Qualifying Index table or Qualifying Code table, identified in the program listing as **qual_ic** and **qual_cc**, respectively.

In the preferred embodiment described herein, the three values of the *rule_type* field that trigger execution of qualifying logic are "II", "IC", and "CC". "II"-type rules are qualifying rules specific to the Index Code and, for example, may require two or more occurrences of the Index Code in a patient history with different dates of service. "IC"-type rules define criteria for Index Codes relative to procedure (CPT) category codes. An "IC"-type rule identify CPT categories (not specific CPT codes) for the specific Index Code. "CC"-type qualifying rules are similar to "IC" rules, but instead of checking for a certain number of one type of procedure category, the "CC"-type logic checks for a single occurrence of each of two separate procedure categories.

Pertinent portions of the ***qual_check*** function are reproduced below.

```
open mast_curs using in_scope
fetch mast_curs into qm.*
let hold_status = status
while hold_status != notfound
        open grp_crs using qm.rule_group
        fetch grp_curs into qg.*
        while status != notfound
        ...
        when qg.rule_type = "II"
        ...
        when qg.rule_type="IC"
        ...
```

when qg.rule_type="CC"

...

Thus, depending on the value of the *rule_type* field, the program applies one of the sets of qualifying logic to determine whether a patient's record satisfies the appropriate set of qualifying rules for that patient.

Type II Qualifying Rules

The program logic for Type II qualifying rules begins by building a SQL query to check the patient record for a certain number of occurrences of specific codes (ICD-9, CPT, HPCPS or category) or Indicator values. The requisite number of occurrences of codes or Indicator values for the particular rule type is stored in the Number required field (*qg.num_required*) of the Qualifying Group table. Upon execution of the query, and if the requisite number of occurrences is found, the ***qual_check*** considers the patient to have successfully passed the Type II qualifying rules.

More particularly, the Type II program logic builds a SQL query based on values read from the **qual_ic** table using the values of *rule_type* and *rule_id* read from the **qual_group** table. If the *cat_cpt* field of the **qual_ic** record is populated (with a category, CPT, HCPCS, or ICD value), the where clause of the SQL statement is expanded to create a statement that checks for a match between the *icd1* field (from the **tmp_index** table) and the value of *cat_cpt*. If *cat_cpt* is not populated, the where clause looks for a match between the *indicator* field in the **tmp_index** table and the value read from the *indicator* field in the **qual_ic** table. This process continues for every record in the **qual_ic** table containing the *rule_type* value read from the **qual_group** table.

When no more records exist in the **qual_ic** table for the given *rule_type*, the SQL statement that was constructed is executed, and the number of records returned is tallied. The total number of records satisfying the SQL query is then compared against the value of the *num_required* field from the **qual_group** table. If the total exceeds the

value of the *num_required* field, the rule is identified as having "passed"; if not, the rule is "failed".

Next, the *logical* field from **qual_group** table is read. The *logical* field indicates whether the qualifying rule is inclusive or exclusive in nature. If the value of the *logical* field is "F", the *rule_passed* variable is inverted (that is, if the rule is exclusionary, and the requisite number of occurrences have been found, then rule was not "passed," and vice versa). Once this step is complete, the ***qual_check*** function checks the *rule_passed* value to determine whether to continue checking the patients' records for qualifying circumstances, or stop processing the patients' records and return control to the main program ***pp_comp.4gl***. If the value of *rule_passed* for the patient's record is not "true", the ***qual_check*** program exits and returns the *rule_passed* value back to the section of ***pp_comp.4gl*** code that called this qualifying logic.

Type IC Qualifying Logic

Similar to the Type II qualifying logic, the Type IC logic initially reads a record from the **qual_ic** table using the *rule_type* and *rule_id* values previously retrieved from the **qual_group** table. For each relevant record in the **qual_ic** table, the program counts the number of records in the **temp_qual** table where the *category* field matches the *cat_cpt* field value found on the **qual_ic** record. This count is then compared against the *num_required* field value from **qual_group**. If the count is greater than or equal to *num_required*, the Type IC logic sets the *rule_passed* variable to "true" (and, as was set forth above for the Type II logic, inverts its value where the value of the *logical* field is "F"). The ***qual_check*** function then checks the *rule_passed* value to determine whether to continue checking the patients' records for qualifying circumstances. If the value of *rule_passed* for patient's record is not "true", the ***qual_check*** program exits and the *rule_passed* value is returned the main program.

Type CC Qualifying Logic

The Type CC qualifying logic differs from the Type II and IC logic in that it obtains its qualifying rule information from the **qual_cc** (Qualifying Code) table rather

than **qual_ic** (Qualifying Index) table. For each record in **qual_cc** matching the *rule_type* and *rule_id* from **qual_group** the following steps occurs:

1. The number of records in **temp_qual** where the *category* field matches the value in the *cat_cpt1* field from **qual_cc** is tallied.

2. If this count is greater than or equal to 1, the number of records in **temp_qual** where the *category* field matches the value in the *cat_cpt2* field from **qual_cc** is tallied. If it is not, the Type CC code skips to the logic segment in step 4 (below).

3. If the count is less than the value of the *num_required* field from **qual_group,** the *logical* field from **qual_group** is checked, and if the value of *logical* is "T", the *passed* variable is set to "false". The *passed* variable is also set to "false" if the count is not less than the value of the *num_required* field and the value of *logical* is "F." (If the count is not less than *num_required*, the code skips to the logic in step 4.)

4. If the *passed* variable is false, the section of code exits and passes control back to the area of the program that called this logic; otherwise the program checks for another relevant record in the **qual_cc** table.

5. When no more relevant records exist in **qual_cc**, this section of code exits and returns control back to the area of the program that called this logic, returning the value of the *passed* variable to the main program (as in the Type II and Type IC logic segments).

In each of the aforementioned qualifying logic segments, the ***qual_check*** function evaluates whether the qualifying logic is considered "passed" or "failed." If the rule is considered "failed," then the records for the patient currently being processed have been disqualified for further processing for the selected Index Code. The function continues processing with the next patient. When no more patients remain, the

qual_check function returns control back to the main body of the ***pp_comp.4gl*** program.

a) Categorize Procedure Codes in Patient History

Additionally, at 1645, as part of the ***foreach*** loop that calls the ***qual_check*** function, the program executes the following two statements appearing at the bottom of page 4 and continuing to page 5, which determine categories for the procedures codes appearing in each patient record and append a category code to the patient record:

```
open get_cat using q.cpt
fetch get_cat into q.category
```

The category codes are used by the ***qual_check*** function as part of qualifying patients for episode of care creation, at 1640, and sorting episodes of care into profile classes, at 1680.

b) Use Clear Window To Identify Episodes of Care

After processing each patient history against the applicable qualifying rules, the program, at step 1650, begins to build episodes of care for patient histories that did not fail the qualifying rules. A clear window time period delimits the onset and resolution of an episode of care. The clear window time period is selected for a specific Index Code from the Window Table.

In the ***pp_comp.4gl*** program, the function call on page 6 to ***report r_edit*** begins clear window processing.

```
finish report r_edit
```

The ***report r_edit*** function (appearing on pages 8 and 9 and reproduced in pertinent part below) identifies the proper clear window time period, flags (for later processing) records indicating a medical complication, and then applies the clear window period to identify discrete episodes of care.

```
report r_edit (c, l, i, cur_by)
    output
    ...
```

order by c.patient, c. relationship, c.sex, l.date_of_serv

...

select beg_win into win_max

from window

where staging in

(select staging from index where index = ir.index)

First, ***report r_edit*** function sorts the claim line item records by patient, relationship, sex and date of service. The ***report r_edit*** function then determines the proper clear window period for the selected Index Code (which index corresponds to the ICD-9 codes appearing in the line item records now being processed). The *beg_win* (Beginning Window) field of the **window** (Window) table defines the clear window period, *win_max*, that is, the maximum number of days without the occurrence of a service relating to a given medical condition (Index Code) that defines the beginning of a new episode of care. The ***report r_edit*** function identifies the appropriate record in the Window Table from which to extract the Beginning Window value by matching the Staging values in the Index Table record for the selected Index Code with the Staging Indicator in the Window Table record for the selected Index Code. In the Index Table, each Index Code is associated with a Staging value. In the Window Table, each unique combination of Index Code and Staging Indicator value is associated with a Beginning Window size.

In addition, at 1655, patient records identified with a complicating diagnosis code are tallied (and flagged to be removed from EOC processing later, at step 1660). Specifically, in the following segments of the ***report r_edit*** function (on page 11 of the program listing), each line item for every patient record in the **temp_data** table is checked for ICD-9 codes corresponding to an ICD-9 code having an Indicator value "C" (from the **tmp_index** table) and any such records are flagged.

open cnt_complic using l.icd1

```
fetch cnt_com;lic into ok_flag
close cnt_complic


if ok_flag then
...
        if not cur_eoc_is_bad then
                let eoc_comp = eoc_comp + 1
                let an_eoc_was_bad = true
                let cur_eoc_is_bad = true
                let cur_status = "C"
        end if
end if
```

Following the flagging of complications at 1655, the program then proceeds sequentially through the claim line item records in the **temp_data** table (on a patient-by-patient basis) and identifies whether or not the applicable clear window period has expired between any two consecutive records. This algorithm uses the *win_max* variable that was populated earlier in step 1650 with the proper Beginning Window value for the ICD-9 code on the record. The date of service in each record is compared with the date of service in the record immediately preceding it chronologically. If the two records being compared are separated by a time period equal to, or greater than, the clear window period (*win_max*), the later record is identified as the beginning event of the a new episode of care. This iterative process is continued for all remaining line item records for all patient claims and is implemented by the following segments of the ***report r_edit*** function (appearing on page 11):

```
if l.date_of_serv – prev_dos >= win_max then
        ...
        let eoc_cnt = eoc_cnt + 1
```

```
        let cur_eoc_is_bad = false
        let eoc_cnt_for_pat = eoc_cnt_for_pat + 1
        let cur_eoc_num = cur_eoc_num + 1
        let cur_status = "V"
    end if

    let prev_dos = l.date_of_serv
```

An alternative embodiment, not implemented in the Microfiche Appendix, can employ a second process to delineate potential episodes of care. In such embodiment, the Window table is populated with values in both the Beginning Window and Ending Window fields. The Ending Window defines a post-episode clear window period, which may be different from the pre-episode clear window (Beginning Window). In this manner, an episode of care can be defined relative to asymmetrical clear window time periods.

In the present embodiment, after the program checks that the clear window period has not been exceeded, the claim line item is associated with a potential episode and inserted into the **eoc** table. Once all line items are so processed, the **eoc** table replaces **temp_data** as the repository for all patient claims detail information and is used for all further processing.

c) Remove Patients With Complications

At step 1660, the program removes from further consideration patients having complications in their medical claims history, as indicated by a flag referred to above in step 1655. Namely, all records for patients flagged as having complications are deleted from the **eoc** table. This step is subsumed within the program statements for the ***report r_edit*** function. More particularly, the statement "**put ins_pat_eoc**" inserts the patient, relationship, and sex values for patients identified with complications into a temporary table, **pat_eoc**, as specified in the following code, found on page 9 of the program listing:

```
create temp table pat_eoc (
        patient         char(15),
        relationship    char(1),
        sex             char(1)) in userspace1;

declare ins_pat_eoc cursor for
        insert into pat_eoc values (c.patient, c.relationship, c.sex)
open ins_pat_eoc
```

The following program segment, found on page 6 of the ***pp_comp.4gl*** program listing, deletes every record from the **eoc** table containing a patient, relationship and sex combination listed in the **pat_eoc** table, thus removing all of the records for every patient who was considered as having complications for the stated medical condition:

```
prepare del_comp_eoc from
        "delete from eoc where e_claim_id = ?"

call errorlog ("updating Comp Patients")
declare comp_pat_curs cursor for
        select unique e_claim_id
                from e_claim cc, pat_eoc pe
                where cc.patient = pe.patient and
                        cc.relationship = pe.relationship and
                        cc.sex = pe.sex
```

```
...
foreach comp_pat_curs into c.e_claim_id
...
        execute del_comp_eoc using c.e_claim_id
end foreach
...
call errorlog ("done with comp Patients")
```

Thus, at this step, all records for patients having a complication flagged in their medical claims history are deleted from the **eoc** table and removed from further consideration for episode or profile building.

d) Qualify Episodes of Care For Profile Assignment

At step 1670, each potential episode of care in the **eoc** table is checked against EOC qualifying rules to determine whether the episode will be assigned to a profile. Episodes that fail the qualifying rules are not removed from the **eoc** table; but neither are they assigned a profile. Step 1670 is implemented in pertinent part by a ***foreach*** statement that loops through each record in the **eoc** table, which, as mentioned previously, now contains all claims line item records that have been found to be part of a valid episode of care.

The following statements (including the ***foreach*** statement) appears in the ***pp_comp.4gl*** program listing beginning on page 7:

```
open qual_ins
let icount = 0
foreach qeoc_curs into cur_eoc_num, q.date_of_serv, q.cpt, q.icd1
...
        let q.category = " "
        open get_cat using q.cpt
```

```
        fetch get_cat into q.category

        if icount=0 then
            let prev_eoc = cur_eoc_num
        end if
        ...

        if cur_eoc_num != prev_eoc then
            close qual_ins

            let eoc_profile = " "
            call qual-check("E") returning passed, eoc_profile, rule_err
            execute upd_eoc using eoc_profile, prev_eoc

            ...
            open qual_ins

            let prev_eoc = cur_eoc_num
        end if

        put qual_ins
    end foreach
```

Before invoking the ***foreach*** statement, the program begin by opening a temporary table, **qual_ins**, that is used for storing a patient's records based on the results of the qualification process (that is, the ***qual_check*** function). Thereafter, the

foreach loop is begun. In the ***foreach*** loop, an ***if/else*** conditional is used to determine whether the record being processed is the first patient record in the **eoc** table, and if so, initializes the *prev_eoc* variable to the current EOC number. Thereafter, the ***qual_check*** function is invoked with a value of "E" in the *in_scope* variable, which indicates that episode qualifying rules are to be used by the function.

As is set forth in detail in Section (d) above, the ***qual_check*** function executes different logic based on the type of qualifying rules that are associated with the selected Index Code. For episode qualification, the same three sets of qualifying logic (Type II, Type IC, Type CC) are employed as in the patient qualification process, except that access to the qualifying tables (and rules) is determined by the scope value "E" rather than "P". Again, the qualifying rules are defined by the contents of the same set of four qualifying tables – the Qualifying Master, Qualifying Group, Qualifying Index, and Qualifying Code tables. For episodes of care, however, the qualifying rules determine if a potential EOC meets the minimum profiling criteria expected for the selected Index Code (e.g., episode includes procedure codes indicating surgical services required for the medical condition).

As compared with its operation in the patient qualification process set forth above, when executed for episode qualification, the ***qual_check*** function evaluates whether the qualifying logic only until the first set of rules are "passed." If any rule is considered "passed," then the episode currently being processed has qualified for profiling. The ***qual_check*** function discontinues episode qualification and returns control back to the ***pp_comp.4gl*** program. In addition to the *rule_passed* value, the ***qual_check*** function returns to the main program a value in the *eoc_profile* variable, which profile number (*profile_num*) is then inserted into the **eoc** table. The ***qual_check*** function sets the value of eoc *profile* to equal the contents of the Profile field of the Qualifying Master table (*qm.profile*). If the episode of care does not satisfy the qualifying criteria, the *eoc_profile* variable the episode is not assigned a profile. Thus, the ***qual_check*** function not only determines whether the episode may profiled but also to which profile it belongs.

The profiles assigned to episodes correspond to combinations of treatment patterns that are likely to arise for a given medical condition. There are eight basic

profile classes to which an episode of care may be assigned. The profile classes identify common combinations of treatment patterns that are useful for statistically analyzing and reporting on medical provider billing data. These Profile Classes are:

0. Common Profile (diagnostic and E/M services common to all of the above).
1. Surgery/Medicine/Radiation Profile
2. Medicine/Radiation Profile
3. Surgery/Radiation Profile
4. Surgery/Medicine Profile
5. Radiation Profile
6. Medicine Profile
7. Surgery Profile

e) Append Category Information to the EOCs

After all valid EOCs have been assigned to a profile, processing continues at step 1680 with appending category data to the **eoc** table records. Specifically, at step 1680, all of the CPT codes in the **eoc** table records are categorized using the **cat_file** table created at step 1645. This step involves the re-categorization of all CPT codes but only in the patient records that have been qualified for episode of care creation during the previous program step 1670. The functionality is similar to that in step 1670; the difference being that in step 1680, the category code is appended to the **eoc** table record, whereas in step 1670, the category code is held temporarily in a variable to assist in the EOC profile categorization. (During execution of the ***foreach*** loop of step 1670, the program performs a lookup on the category table based on the procedure code of the medical record in question to assist in the profile categorization of an episode.) In an alternative embodiment, not implemented in the Microfiche Appendix, the **eoc** table with category information appended is then used to populate the procedure and category parameter tables, which store historical billing and statistical information by Index Code.

f) Populate the Procedure and Category Parameter Tables

In the above-referenced alternative embodiment, at step 1685, data from qualified **eoc** table records (that now include category codes) is added to the procedure and category parameter tables. In general, data from all of the episodes of care for each Index Code are inserted into parameter tables to allow for summary statistical profiling.

g) Generate Output

In yet another embodiment, statistical profiles and other analysis of the data from all episodes of care are provided through the generation of output reports, 1690. The output reports may be implemented as an online table look-up or a hard copy report.

It is to be understood that the above-described embodiments are merely illustrative of numerous and varied other embodiments which may constitute applications of the principles of the invention. Such other embodiments may be readily devised by those skilled in the art without departing from the spirit or scope of this invention and it is our intent that they be deemed within the scope of our invention.